FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Zadání bakalářské práce

Název:	Automatická detekce přeložených textů
Student:	Jan Peřina
Vedoucí:	Ing. Karel Klouda, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je prozkoumat možnosti automatického řešení dvou problémů, jejichž vyřešení by usnadnilo identifikovat texty či jejich části, které vznikly překladem originálu dostupného na internetu.

Tyto problémy jsou:
1) Rozpoznání textu nebo jeho části, který vznikl jako překlad a nikoli jako originální autorův text.
2) Formulace dotazů pro internetové vyhledávače, které by umožnily efektivně najít (předem neznámý) originál přeloženého textu.

Pro obě úlohy proveďte rešerši již zkoumaných řešení a případně navrhněte vlastní. Zaměřte se na české texty, které vznikly překladem textů anglických. Vše otestujte na vhodně připraveném datasetu.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 20. ledna 2021 v Praze.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Automated detection of text translations

Jan Peřina

Department of Applied Mathematics
Supervisor: Ing. Karel Klouda, Ph.D.

May 13, 2021

Acknowledgements

I would like to express my deepest gratitude to my supervisor Ing. Karel Klouda PhD. for his guidance, advice, and support. I would also like to thank my faculty for providing me with the computational resources necessary for the experiments described in this thesis. To my dear friend, Radoslav Kondáč, for his useful advice related to grammar. To Anežka Krézková for keeping me positive even in the hard times. And least but not last to my friends and family for their support. Thank you!

Declaration

Citation of this thesis

Peřina, Jan. *Automated detection of text translations.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Tato bakalářská práce zkoumá možnosti detekce přeložených částí textu společně s možnostmi dohledání původu těchto textů na internetu. V práci je zopakován experiment s vybranou metodou pro detekci strojových překladů. Tuto metodu se podařilo vylepšit pomocí jiné podobnostní metriky textu a lemmatizace. Byla ověřena její aplikovatelnost na lidský překlad. Bylo též otestováno několik způsobů transformace takto detekovaných částí textu do dotazu pro webový vyhledávač, za účelem efektivního dohledání jejich originálu.

Klíčová slova detekce přeloženého textu, detekce plagiátu, zpracování přirozeného jazyka, strojové učení, python

Abstract

This bachelor thesis explores the possibilities of detecting a translated portions of a text together with ways of search for the origin of such text on the internet. In this thesis an experiment of chosen method for machine translation detection is reproduced. This method was then improved by utilization a different text similarity metric and lemmatisation. The applicability of this method on human produced translation was tested. And several ways of transforming this way detected texts into search engine queries to effectively find their sources on the internet.

Keywords detection of text translation, plagiarism detection, natural language processing, machine learning, python

Contents

List of Figures

List of Tables

List of Code Examples

Introduction

Ever since the first Machine Translation (MT) algorithms were introduced, the barrier between languages has become noticeably smaller. The performance of machine translation algorithms, which has been improving rapidly over the past years, allows to translate whole documents into a state of understandable and grammatically correct translations which are of a quality comparable (Popel et al. (2020)) to texts produced by professionals in the field. They can, however, be also used for malicious purposes.

Motivation

MT algorithms are being widely used to bridge the gap between languages. They can, for example, translate the text of a web page, so that even users who do not speak the original language can access the information contained in it. This concept allows people from all over the world to conveniently access sources previously unavailable to them, whether in electronic or printed versions. However, the new found high fidelity of translated texts also allowed the spread of content piracy.

The traditional methods of plagiarism detection work by comparing the examined document with several candidates from a given database, looking for semantic, stylistic, or content similarities in both texts. This approach has one big pitfall, the database has to be continuously updated, since new documents are produced daily. Due to the sheer number of documents on the internet, these candidates can be often unavailable. This problem becomes even more tangled when more than one language is considered.

In this thesis, I examine the possibilities for building blocks of a potential method for plagiarism detection without the need to use a database of candidates, while also being able to overcome the barrier created by the translation of text. The belief is that, since the process of plagiarism likely utilizes search engines to obtain information sources, if it is possible to detect portions of the text that are plagiarized, it should be possible to some extent reproduce the search query based on the information contained in them. Delegating the lookup for the text origin to the search engine could improve both the time needed for detection and the chance of finding the original document, since the search engines are optimized for such tasks and index new documents and online sources constantly.

Objectives

This thesis aims to explore the following topics:

- Automatic recognition of parts of Czech texts that have been previously translated from English.
- Search engine query formulation from a portion of text, which would allow to locate its origin on the internet.

In the first chapter, I will describe the state-of-the-art methods for both the detection of translated texts and the text origin search. The next chapter further explains how to detect translated text using back translation. The experiments on both machine and human produced data together with improvements providing better results than the original method. The last chapter focuses on how to transform the text to a search engine query to locate its source.

Chapter 1

State-of-the-art

1.1 Detection of translated text

Since the research on the detection of machine translation is much more advanced compared to the detection of human-produced translations, the initial thought was to explore methods for detection of machine translation and possibly take inspiration from them. At the same time, it is a goal to verify whether or not are these methods sufficient for the detection of human translated texts as well.

I was able to find several publications about the detection of machine translation reporting great results. Kurokawa et al. (2009) presented a method for automatic detection whether the text is translated or not, the main focus was to detect the difference between original and translated text with reported 77% accuracy for sentence level blocks of text. However, I was having trouble fully understanding how to properly transform the data and train the classifier. Aharoni et al. (2014) presented a more straightforward method based on the presence of each set of Part Of Speech (POS) n-grams as well as presence of function words from proprietary collection, which is available only for English language. Both of these methods were also evaluated for translations made from Statistical Machine Translation (SMT) models, which are currently outperformed by Neural Machine Translation (NMT) and thus might not provide meaningful results. At last, I found a rather interesting method, which was evaluated on NMT systems and reports good results. Nguyen-Son et al. (2019) exploits the textual deformation created by the MT and report 75% accuracy and F1-score on French-English texts. This method is practically applicable to any arbitrary pair of languages as long as there is an access to well performing NMT models for these two languages, since this method utilizes MT for creation of features based on which the final classification is performed. One interesting thing is that each publication utilizes Support Vector Classifier (SVC), however, I decided to compare several other classifiers to determine if any of them would perform better (see Section 2.4).

1.2 Text origin search

When a document is being tested for potential plagiarism, it is compared to a large number of other documents that are stored in a database, based on various criteria. There are several existing solutions based on this principle, e.g. theses.cz, however they are mostly able to check the plagiarism of documents written in the same language. Ceska et al. (2008) tried to solve the problem of multilingual plagiarism by converting documents into a language-independent form, which may solve the problem of multilingual plagiarism, but there is still a need to store a large number of documents in a database.

To eliminate the need of continuous gathering new documents and cross-analysing them with each other. Nowadays, search engines are used as an entry to the internet, since they index a large portion of it and are able to provide us with relevant information to our questions, sort of like an oracle. Thus, I decided to examine the possibilities of search engines to find the origin of a plagiarized text by mimicking users original search queries, which would solve the need of collecting and indexing new documents daily. Such a method, however, has not been developed yet or at least I did not manage to find any relevant sources.

Chapter 2

Translation detection using back translation

In this chapter, I will explain what is the process of back translation, its implementation in Python programming language, and the way I reproduced the experiment to test whether or not it is applicable to the Czech language, based on my observations.

2.1 Method description

The main idea behind using back translation (Nguyen-Son et al. (2019)) to detect translated texts is to simulate the deformation produced by MT systems.

When a MT system translates a text, even if the result is grammatically correct, it shows signs of its artificial origin. This fingerprint is highly perceptible on sentences produced by older machine translation systems and is being reduced over time by the improvement of machine translation methods. After the introduction of deep learning into the field of machine translation, NMT took over, the quality of the produced texts has improved rapidly compared to the previous state-of-the-the-art methods of SMT, however, they are still not perfect.

Back translation is a process where a text is translated from the source language into the target language and then back to the original one. Back translating a text that has been already translated often produces less varying output compared to the author's own text. This can be compared to the process of equalizing the histogram of an image. If the image was already equalized, the difference would have been less significant than in the original image.

The portions of the text, whether sentences or paragraphs, are translated from a source language into an intermediate language, and then back to the source language. This, during seven cycles, produces texts in the source language against which the amount of variance is measured. The amount of variance is measured using BLEU score (see Section 2.1.1) between each sentence and its direct back translation, this whole process is shown in Example 2.1.1. The produced feature vectors, containing these BLEU scores, are then used for the classification task.

Example 2.1.1 (Feature vector creation)

Fisrt iteration		
Original sentence (c):	*Dnes je venku velice pěkné počasí.*	*BLEU*
English translation (e_0):	*It's a very nice weather out there today.*	≈ 0.097
Back translation (b_0):	*Dneska je moc hezké počasí.*	(c, b_0)
Second iteration		
Back translation (b_0):	*Dneska je moc hezké počasí.*	*BLEU*
English translation (e_1):	*It's very nice weather today.*	≈ 0.127
Back translation (b_1):	*Dneska je moc pěkné počasí.*	(b_0, b_1)
Third iteration		
Back translation (b_1):	*Dneska je moc pěkné počasí.*	*BLEU*
English translation (e_2):	*It's very nice weather today.*	*1.0*
Back translation (b_2):	*Dneska je moc pěkné počasí.*	(b_1, b_2)
Fourth iteration		
Back translation (b_2):	*Dneska je moc pěkné počasí.*	*BLEU*
English translation (e_3):	*It's very nice weather today.*	*1.0*
Back translation (b_3):	*Dneska je moc pěkné počasí.*	(b_2, b_3)
$\vdots$		
Seventh iteration		
Back translation (b_5):	*Dneska je moc pěkné počasí.*	*BLEU*
English translation (e_6):	*It's very nice weather today.*	*1.0*
Back translation (b_6):	*Dneska je moc pěkné počasí.*	(b_5, b_6)

Score vector: $(0.097, 0.127, 1.0, 1.0, 1.0, 1.0, 1.0)$

In Example 2.1.1, it is shown how the feature vector for a sentence is created. At the beginning, a Czech sentence is translated into English and then back to Czech, the similarity between those two sentences is measured and then the back translated sentence is used in the next iteration. After just two iterations, the sentence is in a state when its following back translations are the same, so the score is always 1 from this point.

The reported results for this method were 75% on both accuracy and F1-score, which is a solid result, however, these results were measured on French-English parallel text and it was not clear whether or not this method will obtain at least as good results on Czech-English parallel texts as on the original one.

2.1.1 BLEU score

The BLEU score was originally developed by Papineni et al. (2002) to automatically evaluate the closeness between a translation produced by a MT system, and human translation posing as reference. It was stated that BLEU score highly correlates[1] with human judgement over the quality of machine translated texts. It uses a modified n-gram precision which is calculated as the number of n-grams of the translation contained in the reference, limited by the number of occurrences of n-gram inside of the said reference, divided by the length of the translation. First, the geometric mean of the modified n-gram precision for n-grams p_i for $i = 1, 2, 3, 4$ is calculated. The weights $w_1, \dots, w_4$ by default uniformly distributed are introduced, so it is possible to customize how much each n-grams contributes to the BLEU score. The brevity penalty (BP)

$$BP = \begin{cases} 1, & \text{if t > r} \\ e^{\left(1-\frac{r}{t}\right)}, & \text{otherwise} \end{cases},$$

is uded to penalize translations shorter than reference. The t represents the length (number of words) of translation and r representing the length of the reference. The BLEU score is then defined as

$$BLEU = BP \cdot \exp\left(\sum_{i=1}^{N} w_i \log p_i\right).$$

Demonstration of the BLEU score calculation can be seen in Example 2.1.2.

[1] with 0.96 correlation coefficient

Example 2.1.2 (BLEU score calculation)
Reference: Dnes je velice pěkné počasí.
Translation: Dnes je velice krásné počasí.
Modified precisions

type	*matching*	p_i *(matching/total)*
Unigrams	*Dnes, je, velice, počasí*	*(4/5)*
Bigrams	*Dnes je, je velice*	*(2/4)*
Trigrams	*Dnes je velice*	*(1/3)*
Quadgrams	*(none)*	*(0/2)*

Brevity penalty

type	*length*
reference	*5 = r*
translation	*5 = t*

$$BP = e^{1-\frac{r}{t}} = e^{1-\frac{5}{5}} = e^{1-1} = e^{0} = 1$$

$$BLEU = BP \cdot \exp\left(\sum_{i=1}^{N} w_i \log p_i\right) =$$

$$= 1 \cdot \exp\left(0.25 \cdot \log\frac{4}{5} + 0.25 \cdot \log\frac{2}{4} + 0.25 \cdot \log\frac{1}{3} + 0.25 \cdot \log\frac{0}{2}\right) \approx$$

$$\approx 0.427$$

2.2 Implementation

To verify whether this method performs well on the Czech language or not, I had to implement a data processing pipeline that would transform the data, in this case English sentences of various lengths and their Czech translations, into seven-dimensional vectors containing the BLEU scores of back translations, the same way as it was described above.

2.2.1 Translation

The first major component is the ability to translate text from the Czech language into the English and vice versa. My initial thought was to back translate the dataset using the Google Translate[2] service and, although it worked well when tried on a few samples, it seemed rather unpractical to do manually. This could be of course automatized, however, I would be certainly blocked or at least restricted after several requests to the service, since it is against the terms of use of the service[3].

[2]`https://translate.google.com`
[3]`https://policies.google.com/terms`

The next strategy was to utilize translation services, however, the leading suppliers of such services demanded to be paid per translated character which would be expensive in this case.

The last and final way was to obtain a set of machine translation algorithms between Czech and English on my own. Fortunately, thanks to Tiedemann & Thottingal (2020) I was able to obtain a pair of NMT models for translation between the two languages. The "Transformers" module can load a pretrained model either from a local filesystem or from a remote repository. To do so, the user can simply enter the path or identification string of a specific model and the module then downloads it. Thanks to this and the format of the OPUS-MT model identification strings being "Helsinki-NLP/opus-mt-{src}-{dest}" with *src* as the short code[4] for source language and *dest* as the short code for the target language, I was able to obtain the two final Marian-MT (Junczys-Dowmunt et al. (2018)) models.

To further simplify this process, I created a Python function with parameters *src* and *dest* with which I then formatted the identification string of the model, downloaded the model, and returned a function which takes a string in the source language and translates it with it. This procedure is shown in Code 2.1.

[4] "cs" for Czech, "en" for English

```
def create_translation_function(src, dest):
    # model identifier
    model_name = f"Helsinki-NLP/opus-mt-{src}-{dest}"

    tokenizer = MarianTokenizer.from_pretrained(
        model_name, cache_dir=os.environ.get("TRANSFORMERS_CACHE")
    )
    model = MarianMTModel.from_pretrained(
        model_name, cache_dir=os.environ.get("TRANSFORMERS_CACHE")
    )
    # nested function for the translation itself
    def translate(text):
        encoded_translation = model.generate(
            **tokenizer.prepare_seq2seq_batch(
            text, return_tensors="pt"
            )
        )
        translations = [
            tokenizer.decode(t, skip_special_tokens=True)
            for t in encoded_translation
        ]
        return " ".join(translations)

    return translate

translate_to_czech = create_translation_function('en', 'cs')
translate_to_czech('Good morning.')
>> 'Dobré ráno.'
```

Code 2.1: Translation function factory with example

2.2.2 Back translation function

Following the pattern in Nguyen-Son et al. (2019), this function processes the dataset (further described in Section 2.3) in such a way that Czech part of the corpus is treated as the original (class 0) and the English one as the translation (class 1). The English texts are translated into Czech language to simulate real life scenarios, and for each entry the BLEU scores for back translations are produced. An example of the function implemented in Python is shown in Code 2.2.

2.2.3 Casefolding

Another step, which was also mentioned in Papineni et al. (2002), was the case folding before the BLEU score calculation. Case-folding is a transformation of a string either into its lowercase or uppercase form. This way, it is effortless to match words with different casings which were produced either by capitalizing every letter of a word, probably to highlight some important information in the text, or just the first characters of words at the beginning of sentences.

Some case-folding algorithms, for example the one implemented in Python strings, go further and convert some language special characters into a universal form, e.g., German "ß" is converted into "ss" as described in Python Software Foundation (2021), however, a simple lower casing should be also sufficient, since the occurrence of such characters in English and Czech texts is negligible.

2.2.4 Lemmatisation

The final step in the process, which was my own idea for the improvement of matching n-grams to further improve the quality of the BLEU metric, was the lemmatisation of the words. Lemmatisation is a procedure in which inflected word forms such as "running" or "ran" are converted into their base form "run". Lemmatisation process is crucial for texts in Czech language, since it is morphologically richer than English, thus the reduction of the words to the normal form is more beneficial and it is more likely to get better and more accurate score results. For the process of lemmatisation I used Šmerk & Rychlý (2009).

The only problem is that lemmatisation does not solve the issues with synonyms. When comparing the sentences and translations from 2.2, it is worth notice that Czech words "Lék", "Přípravek" and "Výrobek" represent the same thing, but are indeed completely different words and thus cannot be matched. Resolving this could help to produce better results, however, I decided to use only lemmatisation on account of verifying this method in the Czech language.

```
def back_translate(text, translated, score_calculator=BLEU_score) -> dict:
    # translate English text into Czech if flag is True
    if translated: text = en_to_cs(text)

    czech = [text] # list of Czech texts

    for x in range(7):
        czech.append(en_to_cs(cs_to_en(czech[x]))) # back translation

    score = [score_calculator(czech[x], czech[x + 1]) for x in range(7)]

    return {
        "y": [int(translated)],
        "sentence": [text],
        **{f"score_{x}": [score[x]] for x in range(7)},
        **{f"translation_{x}": [czech[x]] for x in range(7)},
    }
>>> back_translate('Lék je určený výhradně k vnitřnímu užití.')
{'y': [0],
 'sentence': ['Lék je určený výhradně k vnitřnímu užití.'],
 'score_0': [0.2626909894424158],
 'score_1': [0.145357684224205484],
 'score_2': [1.0],
 'score_3': [1.0],
 'score_4': [1.0],
 'score_5': [1.0],
 'score_6': [1.0],
 'translation_0': ['Přípravek je určen výhradně k vnitřnímu použití.'],
 'translation_1': ['Výrobek je určen pouze pro interní použití.'],
 'translation_2': ['Výrobek je určen pouze pro interní použití.'],
 'translation_3': ['Výrobek je určen pouze pro interní použití.'],
 'translation_4': ['Výrobek je určen pouze pro interní použití.'],
 'translation_5': ['Výrobek je určen pouze pro interní použití.'],
 'translation_6': ['Výrobek je určen pouze pro interní použití.']}
```

Code 2.2: Function for dataset creation from Czech and English texts

2.3 Dataset creation

In the initial attempt of reproducing this experiment with the Czech language, I used the European Medicines Agency (EMEA) parallel corpus from the Tiedemann (2012), which was created by scraping text from documents and website contents of the agency[5]. This corpus contained 322902 Czech-English aligned documents. The content of those texts varied widely from a few words, mostly upper case names of drugs, to several sentences in one record.

I decided to filter out records containing only a handful of words, since even a professional human translator would have a hard time deciding whether a short sentence, i.e., "Shake before use.", is a machine translation or not, without further context. To obtain dataset with more comprehensive texts, I kept only records with at least forty, but also less than a hundred tokens[6]. This range covers everything from longer sentences to paragraphs. From this point

[5]https://www.ema.europa.eu/en
[6]individual words or terms

onward, I created two balanced versions of this dataset, one smaller, containing 6000 entries and the second one with around 14500 entries, both with a balanced representation between English (class 1) texts and their translated Czech (class 0) equivalents.

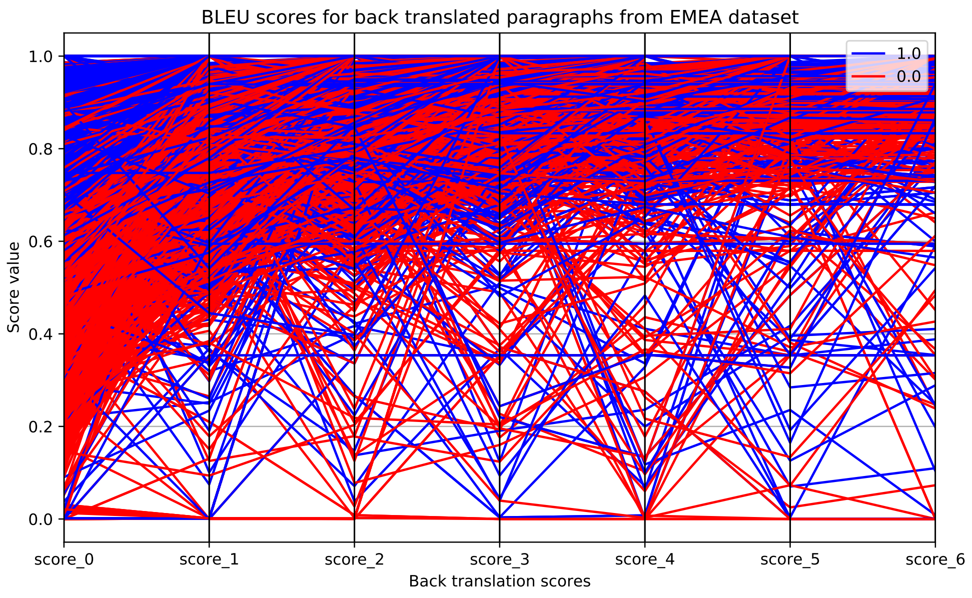

Figure 2.1: Visualisation of the BLEU scores for the smaller EMEA dataset

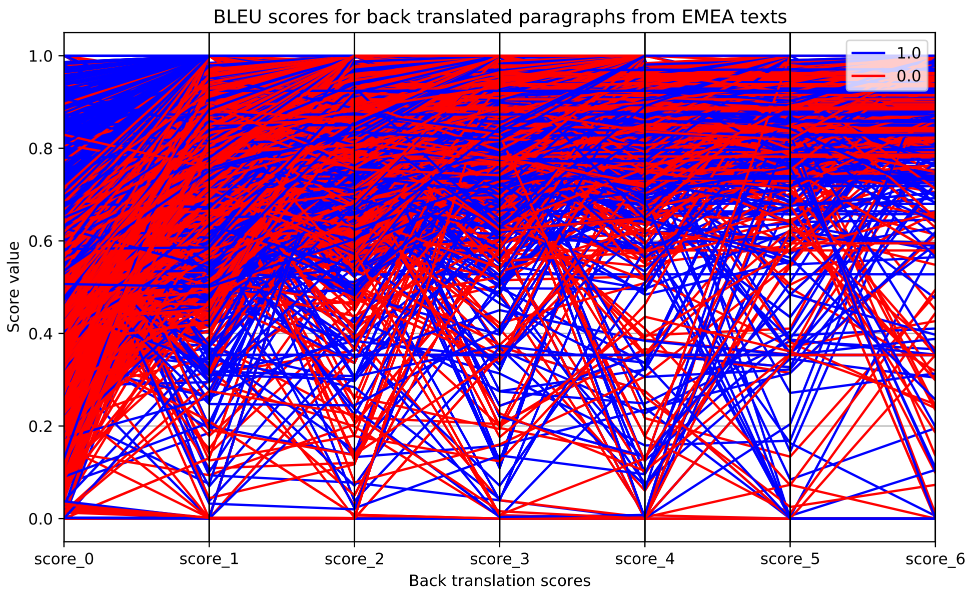

Figure 2.2: Visualisation of the BLEU scores for the larger EMEA dataset

In Figure 2.1 is displayed the processed EMEA dataset, respectively, the BLEU scores for each sample, where blue lines represent the English texts and the red lines represent the Czech translations in the form of a parallel coordinates plot, the same representation is then used for the following visualisations as well. There is a visible trend of a thinning red area starting between 0.01 and 0.7 for the first score and then gradually shrinking towards a higher value for each subsequent score. In comparison, the blue lines tend to attain a higher score from the beginning. With each following score, both trends start to visually merge. Between consecutive scores, there is a deviation of the form of a tooth, the score significantly drops in the value and then again significantly rises in the next step.

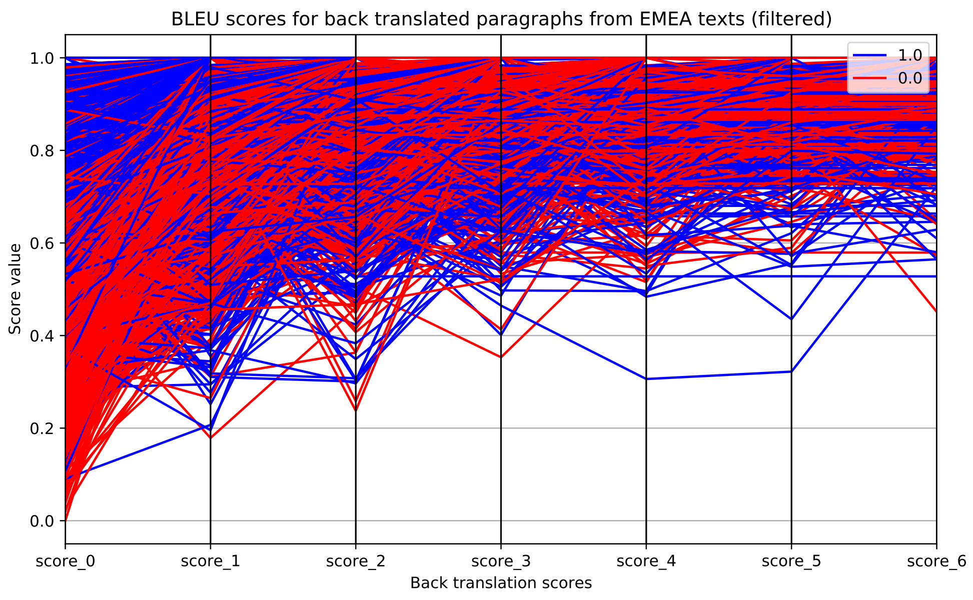

Figure 2.3: Filtered EMEA dataset visualisation

I decided to further inspect those deviations, since they could indicate the presence of a poorly structured text or an error from when the dataset was scraped and created. I assumed that these deviations will be probably more common for earlier iterations, since the translations, since the translations had to stabilize, so I started manually analysing the scores and texts they were produced from, starting from the last one. I found out that the majority of the deviations are indeed insufficient texts containing patterns, e.g., contact information, reoccurring short descriptions of medicaments or their dosage, probably originating from tables, headers or footers from the documents from which the original dataset was created. After discovering these patterns, I decided to filter out these deviations using regular expressions with which I checked for the presence of a specific substring: e. g., “Tel”, or the number of their occurrences, for example I filtered out texts containing specific substrings too many times. By doing this, I effectively filtered out the useless parts of the dataset that could later on bias the performance of the models. The scores for this filtered dataset can be seen in Figure 2.3.

Since it was obvious that the Czech texts were literal translations, produced probably by a MT system, I decided to enlarge the dataset collection with a bilingual version of the book "Alice in the wonderland", which contained considerably less literall translations compared to the EMEA dataset. These texts, however, had to be webscraped from Carroll (1865), but it was not a complex task since the content of the website was static and I was able to iterate over the book chapters thanks to the number of chapters being part of the website address. The book consists of 13 chapters, each containing multiple paragraphs.

To download the paragraphs of each chapter, I performed a Hypertext Transport Protocol (HTTP) request via a very popular Python module called "requests" (Foundation (2005)). This way I obtained the binary data, representing the content of the page written in Hypertext Markup Language (HTML), encoded as bytes. This content had to be decoded and parsed into a data structure that would allow me to retrieve the parallel texts. For this I used the "lxml" (lxml (2005)) Python module that can convert HTML text into a tree structure containing the HTML tags, together with their attributes and contents, while preserving the original hierarchy.

By further analysis of the website structure, I was able to discover that equivalent paragraphs are encapsulated inside a HTML pair tag containing the class "row" inside which were two paragraphs, one for the Czech language and the second one for English. I used the tree structure representing the HTML page and performed a recursive search for such elements using the XPath (Clark & DeRose (1999)) query language to obtain the texts themselves and keep only those containing exactly two paragraphs.

The majority of those paragraph pairs were, in terms of their content, satisfactory. However, there were few pairs, where either both were blank, filled with non-printable characters or their escaped form[7]. Such paragraphs were filtered out either instantly, by the length condition, or after text normalization that converted the escaped symbols into their true forms. The very last thing that had to be done was the removal of the poem at the beginning of the English text. Since there was no text in the equivalent Czech paragraphs, I filtered out such pairs where at least one paragraph tag was empty, to eventually also remove such cases in the rest of the text. The dataset was perfectly balanced and its visualisation of the BLEU scores in Figure 2.4.

[7]"$\backslash n$", "$\backslash t$", etc.

```python
html_parser = lxml.etree.HTMLParser()
normalize = lambda x: unicodedata.normalize('NFKD', x)
total = 0
texts = {'cs':[], 'en':[]}

for i in range(1, 13):
    book_uri = f'http://bilinguis.com/book/alice/cs/en/c{i}'
    response = requests.get(book_uri)

    html = etree.fromstring(response.content, parser=html_parser)
    paragraphs = html.xpath("./div[@class='row']") # recursive query
    ctr = 0
    for div in filter(lambda x: len(x)==2, paragraphs):
        cs, en = div.getchildren()
        if cs is None or en is None:
            continue
        cs_text = normalize(' '.join(cs.itertext()))
        en_text = normalize(' '.join(en.itertext()))

        if not cs_text or len(cs_text) < 30
            or not en_text or len(en_text) < 30:
            continue
        ctr += 1

        texts['cs'].append(cs_text)
        texts['en'].append(en_text)
    total += ctr
    print(f'Chapter: {i} - paragraphs: {ctr}')
print(f'Paragraphs total: {total}')
```

Code 2.3: Web-Scraping procedure of Alice's Adventures in Wonderland Czech-English bilingual book

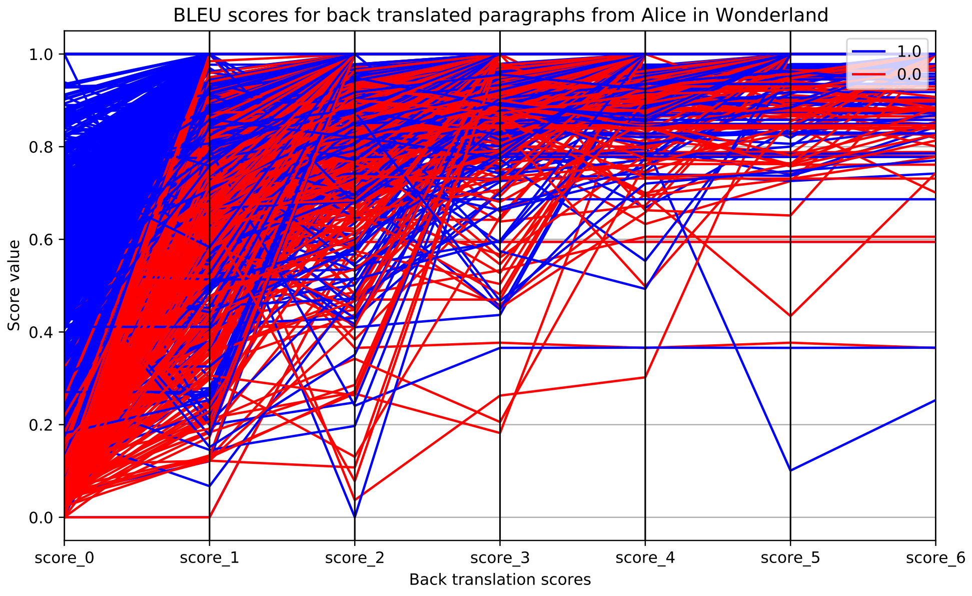

Figure 2.4: Alice's Adventures in Wonderland dataset visualisation

2.4 Evaluation

In this chapter, I will describe the process of dataset evaluation using machine learning models. The main goal is to be able to classify whether the text is original (class 1) or translated (class 0). In the first subsection, I will describe the initial evaluation on a smaller dataset made from the EMEA parallel corpus and in the following subsection, I will compare the results for a larger dataset made from EMEA corpus and a dataset made from the Alice's Adventures in Wonderland book.

2.4.1 Initial evaluation

In the beginning, I decided to classify the datasets using several machine learning algorithms, namely, Ridge Classifier, Stochastic Gradient Descent (SGD) Classifier, SVC, AdaBoost Classifier, and Random Forest Classifier. Each one from the Python module named Scikit-learn (Pedregosa et al. (2011)). For these models, I split the smaller EMEA dataset from the beginning of Section 2.3 into two into two sets, one for the process of training and the other one for testing. Each model was then trained on a train set to learn the relations between the classes and features and then evaluated for accuracy, F1-score, precision, and recall for both train and test sets. This procedure was applied in the same manner in all other experiments. A sample record is shown in Example 2.4.1.

After the initial evaluation on the smaller dataset, I obtained the results shown in Table 2.1. The table contains comparisons of the metrics of binary classification. The table contains scores for each metric as columns, grouped for each set, represented as a percentage rounded to one decimal place. From the contents of the table, it can also be seen that for the training set, the dominant model in terms of the measured metrics was Random Forest Classifier, which was also prominent in the test set, however, it was outperformed by AdaBoost Classifier, with respect to F1-score and Recall. The classifiers altogether predicted with slightly higher precision and F1-score than was reported in Nguyen-Son et al. (2019) for rich-resource languages (75.0% for French). This may be attributed to favorable training dataset, the properties of the Czech language, the lemmatisation, or the combination of the aforementioned. However, these results are still more than satisfactory.

2.4.2 EMEA and Alice in Wonderland

To better verify the applicability of this method on Czech texts, I performed the same measurement on the bigger EMEA dataset with the outliers filtered out. As can be seen in Table 2.2, the Random Forest Classifier, as in the previous experiment, adjusted itself well to the train set, however, this time it was not able to find such success on the test set, probably due to the overfitting

Example 2.4.1 (Processed English record from EMEA dataset)

original	*It is available as 5 mg, 10 mg, 15 mg and 30 mg tablets, as 10 mg, 15 mg and 30 mg orodispersible tablets (tablets that dissolve in the mouth), as an oral solution (1 mg/ ml) and as a solution for injection (7.5 mg/ ml).*
sentence	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablet dispergovatelných v ústech (tablety, které se rozpouštějí v ústech), perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
translation 0	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablet dispergovatelných v ústech (tablety, které se rozpouštějí v ústech), perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 0	*0.785391*
translation 1	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablety dispergovatelné v ústech, perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 1	*1.0*
translation 2	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablety dispergovatelné v ústech, perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 2	*1.0*
translation 3	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablety dispergovatelné v ústech, perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 3	*1.0*
translation 4	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablety dispergovatelné v ústech, perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 4	*1.0*
translation 5	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablety dispergovatelné v ústech, perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 5	*1.0*
translation 6	*Je dostupný ve formě 5 mg, 10 mg, 15 mg a 30 mg tablet ve formě 10 mg, 15 mg a 30 mg tablety dispergovatelné v ústech, perorálního roztoku (1 mg/ ml) a injekčního roztoku (7, 5 mg/ ml).*
score 6	*1.0*
class	*1*

on the train set. In terms of accuracy for the test set, the best performing classifier proved to be the Ridge classifier with accuracy 73.8%, which also had a good F1-score (72.5%), but was outperformed by AdaBoost classifier with the F1-score of 72.8%. Almost all values for the test set are significantly lower than in the smaller dataset, which could be due to the increase in the size of both train and test sets and thus the possible introduction of more challenging samples.

To get a more realistic idea of how the method would perform on authentic data, I repeated the experiment on the dataset obtained from the Alice's Adventures in Wonderland book. The difference between the two datasets is noticeable that EMEA dataset contain less fluent sentences compared to the other one and the content itself is full of medical terms (see Example 2.4.2).

Example 2.4.2 (Example texts from the datasets)
EMEA: *After a dosage of 2 mg kg a Tmax of 1 h (cats and dogs), a Cmax of 1464 ng ml (cats) and 615 ng ml (dogs), and an AUC of 3128 ng. h ml (cats) and 2180 ng. h ml (dogs) is obtained.*

Alice's Adventures in Wonderland: *Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'*

As can be seen from the Table 2.3, the Random Forest classifier once again performed quite well on the train set and it seems, judging by the similarly high values in accuracy and F1-score. However, the major difference in this case is that the gap between Random Forest classifier and other models became significantly lower. Overall high scores were obtained also for the test set, namely, Ridge Regression classifier and SGD classifier regarding the precision score (100%). The best performing model on the testing set was SVC, which obtained the best Accuracy and F1-score values. Furthermore, SVC was also best according to Nguyen-Son et al. (2019), however, the performance was not tested on texts such as this. Another interesting thing are exceptionally high precision scores (100%) for Ridge classifier, Logistic Regression and SGD.

2.5 Undersampling methods

According to the results, it seems that this method is indeed performing quite well. Nevertheless, I wanted to determine if the model performance can be improved with data prepossessing methods. Since I possessed a big dataset with initially balanced classes that, due to the nature of the method, were condensed. I decided to test the effect of undersampling methods, with which I could filter samples that cause class overlap, which could have possibly led to better classification. I used two widely used undersampling methods, Tomek Links and Condensed Nearest Neighbours (CNN).

2.5.1 Condensed Nearest Neighbours

CNN (Hart (1968)) is an incremental method, in which the starting point is a single-element set made from one sample. Next, by classifying other samples based on the aforementioned first element, samples that were misclassified are then added to the set and then repeated until the next incorrectly classified element is found or until all elements from the training set are in the set. The results of this method depend on the order of the samples in the dataset.

The impact of this method on the classification for the larger EMEA dataset can be seen in Table 2.4. The Accuracy decreased, the F1-score has improved for almost every model.

2.5.2 Tomek Links

Tomek Links (Tomek (1976)) is based on the CNN, and is in some way improvement. Compared to its predecessor CNN, Tomek Links are less prone to the order of samples, since it only considers pairs of elements that are closest to each other, yet from different classes, one of them is then removed to strengthen the border between the two classes. It is possible to remove only samples within a certain class, however, this is used mainly for the balancing of the dataset and since my dataset was balanced, I simply removed samples from both classes to further separate them.

From the results in Table 2.5 it is visible that with respect to the train set, Tomek Links slightly improved F1-score of the Ridge classifier. On the other hand, it seems that the performance of other models slightly decreased. The model with the highest accuracy on test set was SGD (73.5%) and every model improved in recall when compared to Table 2.2, but deteriorated in precision.

2.6 Evaluation with different metrics

Since neither CNN or Tomek Links improved the classification significantly, I decided to try this method with text similarity metrics other than BLEU score. The considered metrics and their impact on the results are described below. I also tested the combination of BLEU score alternatives with both Tomek Links, CNN, and without any while focusing solely on accuracy and F1-score.

2.6.1 NIST

The NIST score (Doddington (2002)) is a similarity metric based on the BLEU metric, which, compared to its predecessor, calculates the information weight of each n-gram based on the binary logarithm of the occurences of the (n-1)-gram $u_1, \ldots, u_{n-1}$ over the number of occurences of the n-gram $u_1, \ldots, u_n$

Table 2.1: Results of smaller EMEA dataset classification

EMEA (smaller) results (%)

Classifier	Train				Test			
	Acc	F1	Prec	Rec	Acc	F1	Prec	Rec
Ridge	78.2	78.6	77.2	80.1	77.9	78.6	76.4	80.9
Logistic Regression	78.3	78.5	77.8	79.1	77.8	78.2	77.1	79.4
SGD	78.0	78.9	75.8	82.2	78.0	79.3	75.4	83.5
SVC	78.6	79.0	77.4	80.6	77.8	78.5	76.4	80.8
AdaBoost	78.6	79.7	75.8	83.9	78.4	**79.8**	75.3	**84.9**
Random Forest	**98.3**	**98.3**	**97.3**	**99.5**	**78.7**	78.8	**78.7**	79.0

Table 2.2: Results of larger EMEA dataset classification

EMEA (larger) results (%)

Classifier	Train				Test			
	Acc	F1	Prec	Rec	Acc	F1	Prec	Rec
Ridge	74.5	73.4	73.3	73.4	**73.8**	72.5	72.5	72.5
Logistic Regression	74.6	73.2	73.8	72.6	73.5	72.0	72.5	71.6
SGD	74.0	71.1	76.0	66.8	73.7	70.7	**75.4**	66.5
SVC	74.6	73.3	73.8	72.8	73.6	72.0	72.8	71.2
AdaBoost	74.7	74.2	72.4	76.1	73.4	**72.8**	71.2	**74.4**
Random Forest	**99.1**	**99.1**	**98.6**	**99.6**	71.9	69.8	71.5	68.1

Table 2.3: Results of Alice's Adventures in Wonderland classification

Alice's Adventures in Wonderland results (%)

Classifier	Train				Test			
	Acc	F1	Prec	Rec	Acc	F1	Prec	Rec
Ridge	92.2	91.7	99.5	85.0	93.8	93.2	**100**	87.2
Logistic Regression	94.6	94.4	99.3	89.9	95.8	95.5	**100**	91.3
SGD	96.4	96.3	98.2	94.5	**97.8**	**97.7**	**100**	**95.4**
SVC	95.9	95.9	98.2	93.7	96.8	96.6	99.5	93.9
AdaBoost	97.4	97.5	98.1	96.8	94.8	94.6	95.8	93.4
Random Forest	**99.7**	**99.7**	**99.4**	**100**	96.0	95.9	96.9	94.9

Table 2.4: Impact of CNN on classification of EMEA larger dataset

EMEA (larger) results (%) with CNN

Classifier	Train				Test			
	Acc	F1	Prec	Rec	Acc	F1	Prec	Rec
Ridge	72.6	81.7	74.9	90.0	68.2	73.1	61.3	90.6
Logistic Regression	72.8	81.8	75.2	89.8	68.5	**73.2**	61.6	90.2
SGD	62.0	64.9	87.6	51.6	69.8	61.0	**79.5**	49.5
SVC	73.1	82.2	74.9	91.2	68.1	**73.2**	61.1	**91.2**
AdaBoost	73.5	81.2	78.6	84.0	**70.9**	73.1	65.4	82.8
RandomForest	**99.1**	**99.4**	**98.9**	**99.8**	69.5	70.9	64.9	78.1

itself.

$$\text{Info}(u_1, \ldots, u_n) = \log_2 \left(\frac{\#\text{ of occurences of } u_1, \ldots, u_{n-1}}{\#\text{ of occurences of } u_1, \ldots, u_n} \right).$$

This way the added information weights add more value to less frequently occurring n-grams, which in most cases play a crucial part in determining whether the sentence is a translation or not. The NISt score is defined as

$$\text{NIST} = \sum_{n=1}^{N} \left(\frac{\sum\limits_{\substack{\forall \text{ co-occuring} \\ \text{n-grams } u_1,\ldots,u_n}} \text{Info}(u_1 \ldots u_n)}{\sum\limits_{\substack{\forall \text{ n-grams } v_1,\ldots,v_n \\ \text{in translation}}} 1} \right) \cdot \exp\left(\frac{\log^2 [\min(\frac{t}{r}, 1)]}{2} \right).$$

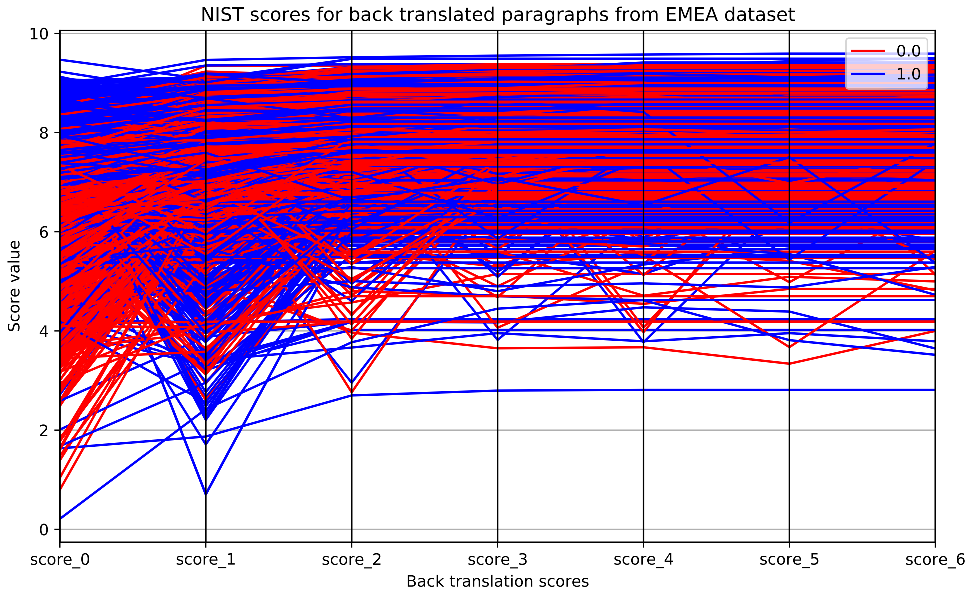

Figure 2.5: Filtered EMEA dataset with NIST score visualisation

When comparing the visualisation of NIST scores (Figures 2.5 and 2.6) to the visualizations of the BLEU scores (Figures 2.3 and 2.4). It appears that the score values are more spread for earlier iterations, and later on the majority of the scores seem to be constant. There are again some visible deviations, a massive one for English texts in the second iteration of the score and a tiny one for the Czech texts in the third iteration. This could indicate similar unusable data that have been filtered out in Section 2.3, or maybe even some hidden relations, however it was not further analysed. The results in the Tables 2.6 and 2.7 consist of three main column groups, the group which

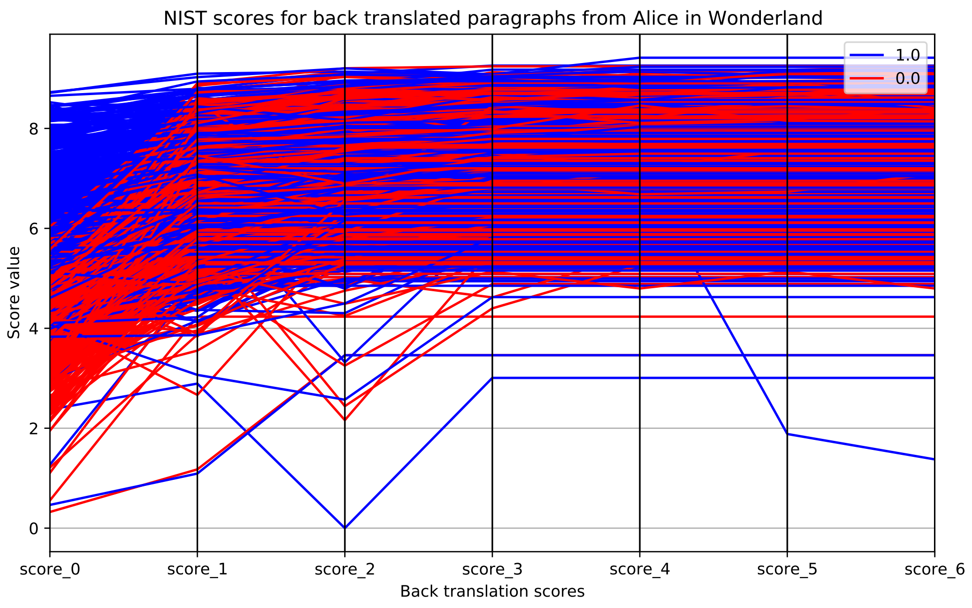

Figure 2.6: Alice's Adventures in Wonderland dataset with NIST score visualisation

represents the use of no undersampling method, Tomek Links and CNN, for each accuracy and F1-score are shown. Based on the results, it seems that this method was able to increase both the accuracy and F1-score of almost every model when used compared to the results of the BLEU score on the larger EMEA dataset (Table 2.2). However, for the dataset of Alice's Adventures in Wonderland, the results with NIST were mostly similar to the results with BLEU in Table 2.3.

2.6.2 GLEU

The GLEU score (Wu et al. (2016)), which was also inspired by the BLEU score, solves some issues of its predecessor. In particular, the GLEU score is better applicable for sentence-level comparisons, in contrast to BLEU, which was designed for corpus level comparison.

The score is based on the precision and recall of the matched n-grams from the reference and translation. Precision is the number of matching n-grams over the total number of n-grams in reference and recall is computed in the same way but with the translation instead. The final GLEU score is the minimum of precision and recall, thus being between 0 and 1, where a higher value indicates a stronger match. Other benefit of this metric is that it is symmetrical.

The visualisations of the GLEU scores for EMEA dataset and Alice's Adventures in Wonderland can be seen in Figures 2.7 and 2.8. They strongly resemble their counterparts for the BLEU score.

Results of the classification based on GLEU scores are presented in Tables 2.8 and 2.9. These results are by a great margin the best compared to the previous ones. For the EMEA dataset, the most successful model was the AdaBoost classifier with 82.9% accuracy and 81.7% F1-score and for the Alice's Adventures in Wonderland dataset (2.9) with 98.5% accuracy and 98.5% F1-score when either Tomek Links or CNN undersampling the method was used.

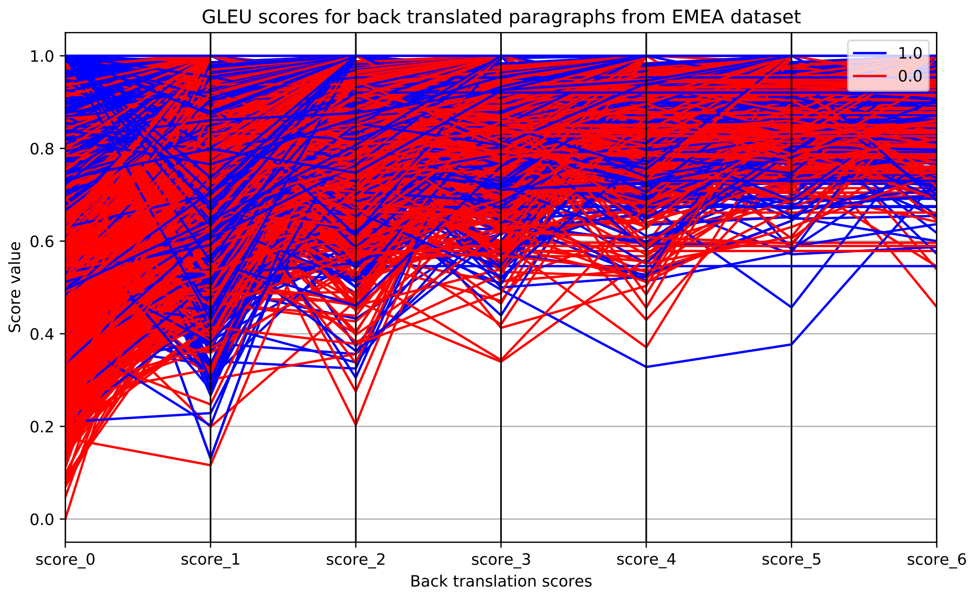

Figure 2.7: Filtered EMEA dataset with GLEU score visualisation

2.7 Results

From the results measured in the previous experiments, it is obvious that this method indeed is suitable for the detection of machine translated texts and works well on the Czech language. The change of the similarity metric allowed me to push both the accuracy and F1-score further, and I believe that those values could be improved even more with some more sophisticated metrics, or maybe, as I mentioned in Section 2.2.4, the ability to match words with the same meaning, for example using word clustering. The maximum accuracy and F1-score for both EMEA and Alice's Adventures in Wonderland datasets were obtained with the GLEU score exceeding the results reported in Nguyen-Son et al. (2019). However, to decide on what classifier to choose is not simple, since there are four candidates, namely SGD, SVC, AdaBoost classifier and Random Forest classifier, which was dominating in at least one classification metric.

Table 2.5: Impact of Tomek Links on classification of EMEA larger dataset

EMEA (larger) results (%) with Tomek Links

Classifier	Train				Test			
	Acc	F1	Prec	Rec	Acc	F1	Prec	Rec
Ridge	76.2	77.6	76.5	78.8	73.3	**73.6**	69.6	**78.1**
Logistic Regression	76.3	77.7	76.8	78.6	73.1	73.4	69.6	77.5
SGD	76.2	76.3	79.7	73.1	**73.5**	72.1	**72.4**	71.9
SVC	76.8	78.2	76.9	79.5	73.1	73.5	69.4	**78.1**
AdaBoost	77.0	78.5	77.0	80.0	72.4	72.8	68.6	77.4
Random Forest	**99.1**	**99.1**	**98.7**	**99.5**	70.9	70.8	67.9	73.9

EMEA (larger) results (%) with NIST score

Classifier	No undersampl.		Tomek Links		CNN	
	Acc	F1	Acc	F1	Acc	F1
Ridge	78.4	77.5	77.5	77.4	68.8	73.9
Logistic regression	78.5	77.1	77.6	76.9	72.0	75.6
SGD	58.8	69.2	56.1	68.0	67.6	73.3
SVC	79.4	76.5	79.3	77.2	74.5	76.8
AdaBoost	79.0	77.3	78.6	77.6	73.1	75.5
Random Forest	**80.7**	**78.6**	**80.2**	**79.0**	**77.1**	**77.7**

Table 2.6: EMEA classification results with NIST score

Alice's Adventures in Wonderland results (%) with NIST score

Classifier	No undersampl.		Tomek Links		CNN	
	Acc	F1	Acc	F1	Acc	F1
Ridge	96.5	96.4	**96.8**	**96.6**	92.8	92.0
Logistic regression	**96.8**	**96.7**	**96.8**	**96.6**	**95.5**	**95.3**
SGD	93.1	92.3	94.0	94.1	91.6	92.0
SVC	96.5	96.4	96.3	96.1	93.5	92.9
AdaBoost	96.3	96.1	95.5	95.3	**95.5**	**95.3**
Random Forest	94.8	94.6	94.5	94.3	93.8	93.4

Table 2.7: EMEA classification results with NIST score

EMEA (larger) results (%) with GLEU score

Classifier	No undersampl.		Tomek Links		CNN	
	Acc	F1	Acc	F1	Acc	F1
Ridge	80.1	78.3	80.2	79.1	76.5	78.9
Logistic regression	80.6	79.1	80.4	79.5	77.4	79.5
SGD	80.6	79.2	79.5	80.1	79.7	80.3
SVC	82.0	80.2	82.3	81.5	76.9	79.1
AdaBoost	**82.9**	**81.7**	82.5	**82.2**	**81.4**	**81.8**
Random Forest	82.7	81.5	**82.6**	82.1	**81.4**	81.5

Table 2.8: EMEA classification results with GLEU score

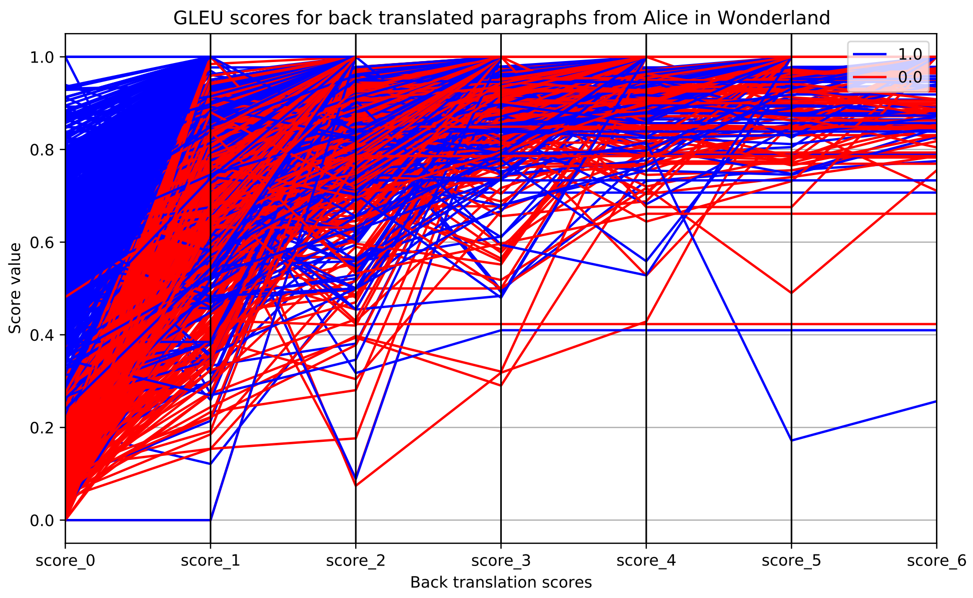

Figure 2.8: Alice's Adventures in Wonderland dataset with GLEU score visualisation

Table 2.9: EMEA classification results with GLEU score

Alice's Adventures in Wonderland results (%) with GLEU score

Classifier	No undersampl.		Tomek Links		CNN	
	Acc	F1	Acc	F1	Acc	F1
Ridge	96.3	96.0	96.0	95.7	95.5	95.2
Logistic regression	97.0	96.8	96.8	96.6	88.6	86.7
SGD	**98.3**	**98.2**	**98.5**	**98.4**	**98.5**	**98.4**
SVC	**98.3**	**98.2**	97.5	97.4	95.8	95.5
AdaBoost	97.3	97.1	96.8	96.6	96.5	96.3
Random Forest	97.3	97.2	96.8	96.7	97.3	97.2

To determine this, I made one last experiment in which I trained the classifiers on the EMEA dataset and then I used Alice's Adventures in Wonderland dataset as a test set to at least slightly simulate the real life scenarios in which the training texts may differ from the evaluated ones. From the results in Table 2.10, it is obvious that SGD was dominant in both accuracy and F1-score. In addition, the undersampling method improved the prediction significantly.

Each classifier was originally trained with default hyper-parameters predefined in the Scikit-learn (Pedregosa et al. (2011)) Python module. For the SGD the Hinge loss (Gentile & Warmuth (1998)) function, L2 penalty, $\alpha = 0.0001$ regularization parameter, and maximum 1000 iterations. However, with hyperparameter tuning I achieved better results. I took a portion of the train set and I then tuned the loss function, α regularization coeffi-

Table 2.10: Results on models trained on EMEA dataset and with Alice's Adventures in Wonderland as test set

Train EMEA, test Alice results (%)

Classifier	Base		Tomek Links		CNN	
	Acc	F1	Acc	F1	Acc	F1
Ridge	67.6	52.1	69.3	55.6	79.1	73.6
Logistic regression	68.7	54.3	71.0	59.1	79.1	73.6
SGD	**69.3**	**55.6**	**75.7**	**67.9**	**82.8**	**79.2**
SVC	63.4	42.2	66.0	48.4	78.7	73.0
AdaBoost	64.7	45.4	67.3	51.4	70.7	58.9
Random Forest	65.4	48.3	67.7	54.0	68.3	54.6

cient, the learning rate, penalty, and maximal number of iterations. With the setup of α=0.000075, "optimal" learning rate, perceptron loss function, and L2 penalty. For a description of the hyperparameters see SGD documentation page. With this setup, I was able to achieve 92.6% on both the accuracy and F1-score. This was further improved, when the training set was filtered using Tomek Links, which filtered out roughly 700 samples from the train set, thus improving the classification. In this case, the hyperparameter setup was α=0.000125, "adaptive" learning rate, perceptron loss function, and L1 penalty and the achieved results were 95% on both accuracy and F1-score on the test set.

2.8 Discussion

The main advantage of this method is that all the building blocks are easily accessible (compared to, e.g., Aharoni et al. (2014)) which allows this method to be widely used.

The results obtained with the method presented in Nguyen-Son et al. (2019) indicate that it is indeed this way possible to detect originally English portions of the text that have been translated to Czech, either by human or a MT model. I think that adding the lemmatisation and cleaning the dataset helped the original method. Moreover, switching the BLEU score for the GLEU score both the accuracy and F1-score improved the result significantly, as well. I believe that introduction of a metric that could match synonyms to solve the issues described in Section 2.2.4 would probably improve this method even more. The final results of the hyper-parameter tuned SGD are indeed interesting and I would like to further inspect the influence of the tuned hyper-parameters on the classification as a follow-up to this thesis, together with more advanced text similarity metric.

It is worth noting that this method could be deceived by human intervention in the form of postediting the translated text, however, this was not examined, since it is beyond the scope of this thesis.

Chapter 3

Text origin search

In this chapter, I will describe my experiments on automatic search of a text origin, for both original and back translated texts, using a search engine, its implementation, and results.

3.1 Method description

Ever since the search engines were introduced, the way mankind used the internet changed drastically. With search engines, people are able to search for information from various sources in a matter of milliseconds and with just a simple textual representation of the topic. With just a handful of words, the search engines are able to recommend relevant results to the query, thanks to continuous gathering information from a significant portion of the internet. These results are also sorted based on the relevancy to the query and overall "popularity" of the page itself.

It is believed that nowadays the majority of topic research is made using search engines. The main goal of this experiment is to try to find an original source of a portion of the text that has been obtained using a search engine, but was translated into Czech language. The idea supporting this approach is that when a person copies a portion of a text found using a search engine, the copied text is probably relevant to the search engine query. Thus, it should be possible to create a search engine query that would allow to search for the origin of the text from it, and in the ideal case the source of the original text would appear on top of the results after performing the query.

In this method, I decided to focus on paragraph-level texts, since I cover the detection of translated texts of the same size in the previous chapter. However, the transformation of the text into a query is a bit more complicated, since numerous methods could be applied to this job. I decided to test three methods: text identity, keyword extraction, and automated text summarization.

3.2 Dataset

To test this method, I created a dataset of ten paragraphs from different sources, such as Wikipedia.org or Medium.com. I chose pages about Robots, Bitcoin cryptocurrency, and ray tracing in graphics. From the Medium.com, I chose two blogs, one tutorial about running 30 prediction models in a few lines of code and the other one about the monetizability of data science skills. Their original sources are recorded in Appendix B. I also added paragraphs from some of the publications (Devlin et al. (2019), Wu et al. (2016), Doddington (2002), Papineni et al. (2002) and Nguyen-Son et al. (2019)), for which I decided not to include their abstracts, since they are usually placed on pages indexing the publications and thus probably indexed by search engines, but rather paragraphs located further lower in the document. All these paragraphs, originally in English, were translated to simulate the process of copying and translating the author's work into Czech and then translated into English to later create and execute the search query.

3.3 Implementation

For the purposes of this experiment, I decided to use Google Search Engine[8] as one of the leading search engines in terms of popularity. However, similarly to the limitations of Google Translate[9] that were mentioned in Section 2.2.1 it is against the terms of use of the Google Search Engine to programmatically execute search queries, but compared to the MT, there is no such thing as a "downloadable" search engine present, since it is costly to maintain such a product in a well-performing state, only solutions were to either pay for the search, or do them manually, which is what I did.

To minimize the amount of exhaustive manual work, I created a function (see Code 3.1) that creates a direct link for the query, so it is not required to manually copy the formulated queries into the Google Search. Initially, since I was collecting 20 results, I was generating two links for each portion of the text, since Google Search by default gives ten results per page. Later, I found out that the number of results can be modified by introducing a HTTP query parameter inside Google Search Uniform Resource Identifier (URI). This helped in most cases, however, for a few of them, I still had to make a click to another results page, since some of them contained additional content, such as images, videos or similar search engine queries, all related to the one I searched for.

After this point, the only thing left to do was to implement text identity, keyword extractor and text summarization functions.

[8]https://www.google.com/

[9]https://translate.google.com/

```
def build_uri(text, transformation=lambda x: x):
    query = transformation(text)

    if isinstance(query, list) or isinstance(query, tuple):
        query = ' '.join(query)
    query =  quote(query, safe='')

    return 'http://google.com/search?q={query}'

build_uri('my search engine query')
>> http://google.com/search?q=my%20search%20engine%20query%26num%3D30
```

Code 3.1: Google Search Egngine URI builder

3.3.1 Text Identity

This transformation was the least demanding one. I simply used the whole paragraph as it was when formulating the query. The reason I wanted to test this method is because when the search engine indexes the original source, it uses the full textual content of the website, so when I use the whole paragraph, I maximize the amount of information originating from the source that can be then used by the search engine to more easily connect the paragraph to its source.

3.3.2 Automated text summarization

With the use of machine learning language models, it is possible to pinpoint important portions of the text, based on the relations obtained by training the deep learning model. This is useful for reducing the larger original text into a smaller one, while preserving most of the relevant information. I was able to do this by using the pretrained Pegasus (Zhang et al. (2019)) model for text summarization obtained from the Transformers (Wolf et al. (2020)) Python module.

3.3.3 Keyword extraction

The main purpose of keyword extraction is to even further decrease the original text by extracting only the most significant words or phrases, for example, by the number of occurrences in the text, their semantic relations, or other criteria. These words are then used as the query itself.

As I mentioned, there are multiple keyword extraction algorithms, so I decided to compare the output produced by some of them to get an idea if they perform similarly. For this, I compared my implementation of a keyword extractor based on word frequency, which also filters out unwanted POS tags, to keep only verbs, adverbs, nouns, adjectives, and unclassifiable ones, and is also able to performs either lemmatisation or stemming of the original text. The other method examined was the keyword extractor from Gensim

(Řehůřek & Sojka (2010)) Python module, which is based on the PageRank (Page et al. (1999)) algorithm and supports the lemmatisation of the keywords. The last method was KeyBERT (Grootendorst (2020)), based on the deep learning text BERT transformer (Devlin et al. (2019)). I compared all of them based on twenty keywords extracted from a single paragraph from the dataset (see Section 3.2) using each method and between each pair of keyword extractors the similarity of the produced keywords was measured using the Jaccard similarity

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

and visualised the results as a heatmap (see Figure 3.1). It is visible that none of the different algorithms had high co-occurrence of the produced keywords, but rather the opposite. The other interesting thing is the influence of lemmatisation and stemming. For the keyword extractor based on the word frequency, with lemmatisation, it produces almost the same result to the one where no additional method was used. However, the use of stemming resulted in a much different set of keywords. On the other hand, the lemmatisation process has a much bigger impact on the keyword extractor implemented in the Gensim module. Since the results vary significantly, I decided to use all methods for the final evaluation.

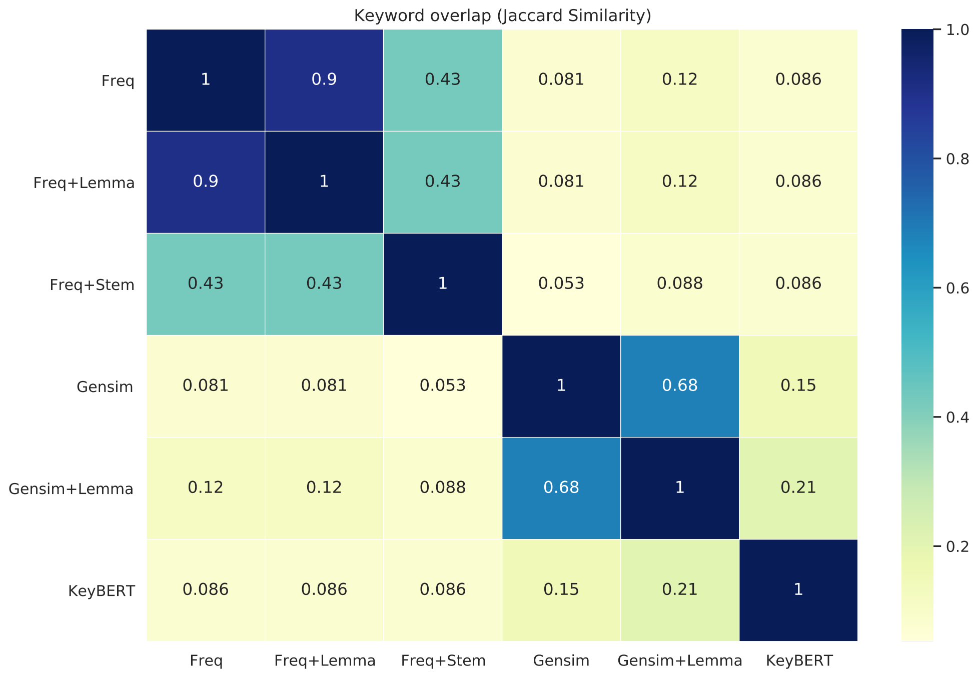

Figure 3.1: Keyword overlap between methods

3.4 URI matching and content mirroring

It is worth mentioning that I encountered two problems when evaluating this approach. The first problem was the mirroring of the document, or its content, on either or completely different sites (see Example 3.4.1), or with a different path (see Example 3.4.2). This lowered the final score, since these URIs were, in fact, false negatives. I examined each of the twenty URIs found for each method-paragraph combination and added newfound links, which represented the document mirror, into a collection of references against which the potential URI was compared.

Example 3.4.1

First archive:	`https://dl.acm.org/doi/10.5555/1289189.1289273`
Mirror 1:	`https://www.researchgate.net/publication/234812513...`
Mirror 2:	`https://www.semanticscholar.org/paper/Automatic-ev...`

For the URI matching itself, I considered only pairs of URI to be matching if they were either the same, or one was prefix of the other. However, this does not solve the issue shown in Example 3.4.2, since the difference between the two URIs is in the centre of the strings. I tried to solve this by thresholding the Jaccard similarity between two sets of strings produced by splitting the original URIs with "/" as a delimiter.

Example 3.4.2

Archive page:	`https://dl.acm.org/doi/10.5555/1289189.1289273`
Direct link:	`https://dl.acm.org/doi/pdf/10.5555/1289189.1289273`

I found this to be very ineffective, since the score was influenced by the URI sizes and thus hard to threshold, but also since it allowed pairs of URIs similar to the one shown in Example 3.4.3 to be matched with a high score, even if they may not be related. Therefore, I ended up using only the combination of exact match and string prefix approach.

Example 3.4.3

```
https://example.domain/standardized_path/Publication_A
https://example.domain/standardized_path/Publication_B
```

3.5 Experiment

To evaluate the results, I used a simple URI relevance score, based on the position of the original URI in the list of results obtained from the search engine. The equation for calculating the score itself is based on the size of results list n, which is at most twenty, and the index of URI i in it, ranging between 1 for the highest relevancy, to 0 for not even being present:

$$\text{relevancy} = \begin{cases} \frac{n-i+1}{n+1}, & \text{if URI is in results,} \\ 0, & \text{otherwise.} \end{cases}$$

This was later enhanced to penalize the methods, for which the query did not result in at least twenty results, which occurred multiple times as can be seen in Figure 3.2. The final score is defined as

$$\text{score} = \frac{n}{k} \cdot \text{relevancy},$$

where k represents the number of results wanted, which was in this case twenty. This is shown in Example 3.5.1. Thus, in an ideal case, if the search engine is able to find at least twenty results based on the query, the penalty vanishes and only the relevancy score is used. This way, I was able to utilize both the order of the results and their number. I collected the results for both the original paragraphs and the back translated ones, to verify that it is possible to find the source using the original text, and if the same thing is possible also with the back translated one.

Example 3.5.1
desired number of results (k) = 20
real number of results (n) = 16
index of the origin url (i) = 3

$$\textit{relevancy} = \frac{n-i+1}{n+1} = \frac{16-3+1}{16+1} = \frac{14}{17} \approx 0.82$$

$$\textit{score} = \frac{n}{k} \cdot \textit{relevancy} = \frac{16}{20} \cdot \frac{14}{17} = \frac{112}{170} \approx 0.66$$

From the Figure 3.2 it can be seen that the queries produced by stemming word frequency keyword extractor together results in less than twenty results frequently. In contrast to the automated text summarization, which produced queries meeting the condition every time. For the rest of the methods, the fulfilment of this condition was slightly worse, each with approximately two queries failing this criteria.

3.6 Results

From Table 3.1, it is noticeable that both the summarizarion and word frequency with stemming did not work well. With automated text summarization as the text transformation method, the Google Search Engine was able to locate the origin of only one of the ten paragraphs. Similarly, the word frequency keyword extractor with stemming was able to find only one original source, however, this time with a lower score 0.35. The best number of sources found total was ten, which was achieved by the text identity, the next best performing being the word frequency keyword extractor, together with Gensim keyword extractor, both were able to find nine of the ten sources total. Next in order were Genism and KeyBERT (eight), then word frequency with lemmatisation (six).

Table 3.1: Results for original paragraphs

Identity	Freq	Freq+L	Freq+S	Gensim	Gensims+L	KeyBERT	Summary
1.0	0.8571	0	0	0	0	0	0
1.0	1.0	0.7	0.35	0.9048	0.6533	1.0	0
0.35	0.9524	0	0	1.0	0.55	1.0	1.0
1.0	0.75	1.0	0	1.0	1.0	0.9524	0
1.0	1.0	1.0	0	1.0	1.0	1.0	0
1.0	0	0	0	1.0	1.0	0	0
0.8095	0.8571	1.0	0	0.9524	0.9524	0.954	0
0.9524	1.0	1.0	0	1.0	1.0	1.0	0
0.9	0.9048	0	0	0.9048	0.5571	0.9048	0
1.0	1.0	1.0	0	0	1.0	1.0	0

The results for multiple methods of query formulation from the original paragraph seemed successful. The results of the back translated equivalents (see Table 3.2), as expected, were not so great. In most cases, the search engine was not able to find the original sources, and when the sources were found, the score was lower on average, which was expected. The best performing methods were in this case, Gensim keyword extractor with lemmatisation (mean 0.290), word frequency keyword extractor (mean 0.289), and KeyBERT (mean 0.240).

Table 3.2: Results for back translated paragraphs

Identity	Freq	Freq+L	Freq+S	Gensim	Gensim+L	KeyBERT	Summary
0	0	0	0	0	0	0	0
0.6	0	0	0	1.0	1.0	1.0	0
0	1.0	0	0	0	0	1.0	0
0.8571	0.0889	0	0	0	0	0.4	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1.0	0	0
0	0.8095	0	0	0	0	0	0.7143
0	1.0	1.0	0	1.0	0	0	0
0	0	0	0	0	0.9048	0	0
0	0	0	0	0	0	0	0

For a better visual comparison of the results, I created two box plots (see Figures 3.3 and 3.4) from the results shown in Tables 3.1 and 3.2.

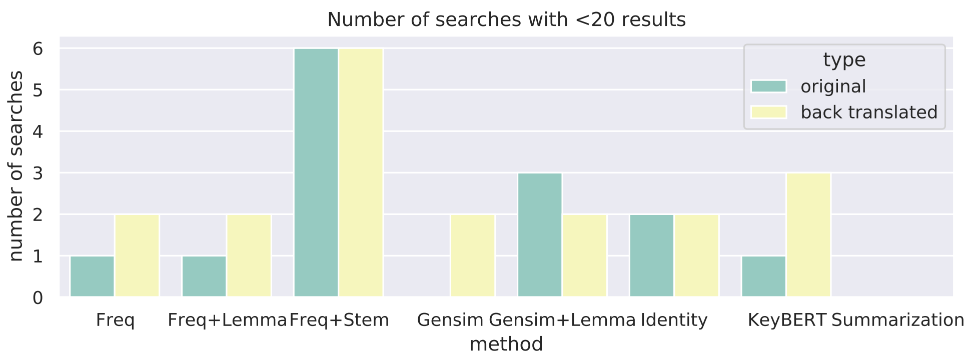

Figure 3.2: Number of search queries resulting in less than twenty results

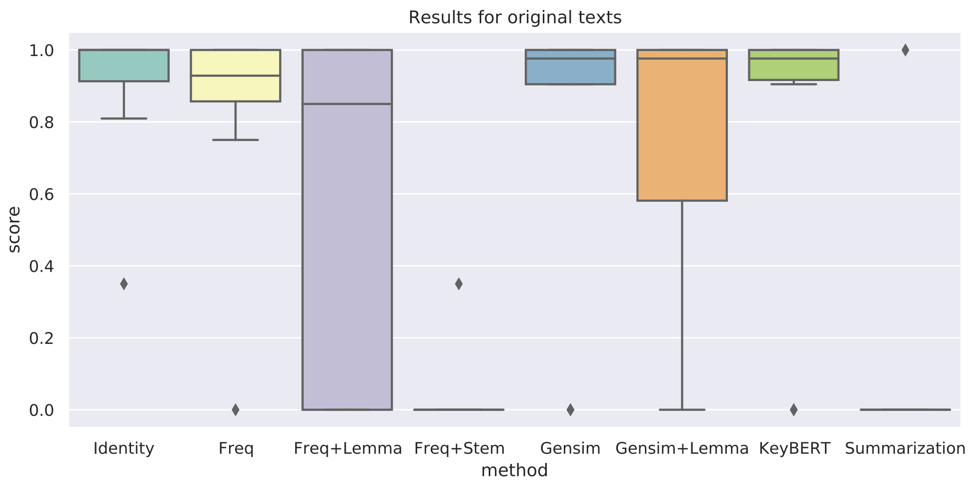

Figure 3.3: Box plot of results for original paragraphs

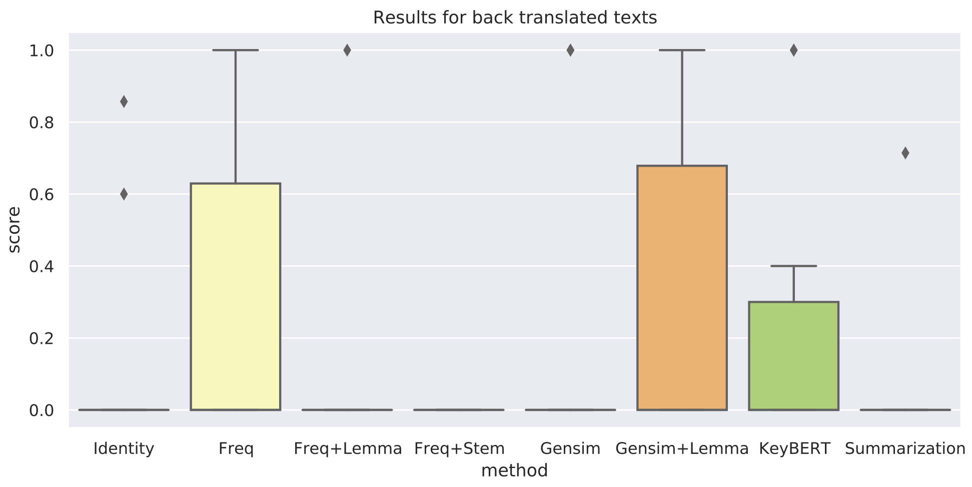

Figure 3.4: Box plot of results for back translated paragraphs

Conclusion

This thesis aimed to examine the possibilities of automatic detection of Czech texts that were created by machine translation from the English originals, and the utilization of search engine queries to find the origin of such texts.

Compared to other state-of-the-art methods (see Section 1.1), the Nguyen-Son et al. (2019) had a straightforward implementation and required no proprietary software or data. I was able to reproduce it, and not only verify its applicability on machine translation, but also improve its results from 69.3% on accuracy and 55.6% on F1-score, up to 95% on both the accuracy and F1-score on the dataset created from Alice's Adventures in Wonderland book and its Czech translation through a combination of lemmatisation, manual cleaning of the training dataset, replacement of the originally used BLEU score (Papineni et al. (2002)) with GLEU score (Wu et al. (2016)), reducing the amount of noisy data with under-sampling methods, and hyper-parameter tuning of the final SGD classifier.

Utilization of the search engine with the goal of finding the original source of the text, in this case the Google Search Engine, did not produce satisfactory results. I examined the influence of using either the whole paragraph, keywords extracted from it using multiple keyword extraction algorithms, and the automatic text summarization, which were then used to build the search engine query itself. By evaluating these methods using a custom score metric, I found out that majority of the original paragraphs were found, however, using the back translated versions of these paragraphs resulted in finding the original source in fewer cases than expected.

In conclusion, I believe that finding a source of translated text through a search engine can be done more effectively, but I was not able to find a right procedure for finding the origin of translated text. Nevertheless, experimenting with this method is time consuming and requires a large amount of manual labour for both creation and validation of the dataset.

Bibliography

Aharoni, R., Koppel, M. & Goldberg, Y. (2014), Automatic detection of machine translated text and translation quality estimation, *in* 'Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)', Association for Computational Linguistics, Baltimore, Maryland, pp. 289–295.
URL: *https://www.aclweb.org/anthology/P14-2048* [Online; Accessed: 24-April-2021]

Carroll, L. (1865), 'Alice's adventures in wonderland'. translator: Císař, Jaroslav.
URL: *http://bilinguis.com/book/alice* [Online; Accessed: 13-February-2021]

Ceska, Z., Toman, M. & Jezek, K. (2008), Multilingual plagiarism detection, *in* D. Dochev, M. Pistore & P. Traverso, eds, 'Artificial Intelligence: Methodology, Systems, and Applications', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 83–92.

Clark, J. & DeRose, S. (1999), XML Path Language (XPath) version 1.0, Recommendation, World Wide Web Consortium.
URL: *http://www.w3.org/TR/xpath.html* [Online; Accessed: 24-April-2021]

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2019), 'Bert: Pre-training of deep bidirectional transformers for language understanding'.
URL: *https://arxiv.org/abs/1810.04805* [Online; Accessed: 24-April-2021]

Doddington, G. (2002), 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, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 138–145.

URL: *https://dl.acm.org/doi/10.5555/1289189.1289273* [Online; Accessed: 24-April-2021]

Foundation, P. S. (2005), ‘Requests: Http for humans™’.
URL: *https://github.com/psf/requests* [Online; Accessed: 24-April-2021]

Gentile, C. & Warmuth, M. K. (1998), Linear hinge loss and average margin, *in* ‘Proceedings of the 11th International Conference on Neural Information Processing Systems’, NIPS’98, MIT Press, Cambridge, MA, USA, p. 225–231.

Grootendorst, M. (2020), ‘Keybert: Minimal keyword extraction with bert.’.
URL: *https://doi.org/10.5281/zenodo.4461265* [Online; Accessed: 24-April-2021]

Hart, P. (1968), ‘The condensed nearest neighbor rule (corresp.)’, *IEEE Transactions on Information Theory* **V**ol. 14(3), 515–516. doi: 10.1109/TIT.1968.1054155.
URL: *https://ieeexplore.ieee.org/document/1054155* [Online; Accessed: 24-April-2021]

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., Seide, F., Germann, U., Fikri Aji, A., Bogoychev, N., Martins, A. F. T. & Birch, A. (2018), Marian: Fast neural machine translation in C++, *in* ‘Proceedings of ACL 2018, System Demonstrations’, Association for Computational Linguistics, Melbourne, Australia, pp. 116–121.
URL: *http://www.aclweb.org/anthology/P18-4020* [Online; Accessed: 24-April-2021]

Kurokawa, D., Goutte, C., Isabelle, P. et al. (2009), ‘Automatic detection of translated text and its impact on machine translation’, *Proceedings of MT-Summit XII* pp. 81–88.
URL: *http://www.cs.cmu.edu/afs/cs/Web/People/dkurokaw/publications/MTS-2009-Kurokawa.pdf* [Online; Accessed: 24-April-2021]

lxml (2005), ‘lxml - xml and html with python’.
URL: *https://lxml.de/* [Online; Accessed: 21-March-2021]

Nguyen-Son, H.-Q., Thao, T. P., Hidano, S. & Kiyomoto, S. (2019), ‘Detecting machine-translated text using back translation’.
URL: *https://arxiv.org/pdf/1910.06558.pdf* [Online; Accessed: 24-April-2021]

Page, L., Brin, S., Motwani, R. & Winograd, T. (1999), The pagerank citation ranking: Bringing order to the web., Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

URL: *http://ilpubs.stanford.edu:8090/422/* [Online; Accessed: 24-April-2021]

Papineni, K., Roukos, S., Ward, T. & Zhu, W.-J. (2002), Bleu: a method for automatic evaluation of machine translation, *in* 'Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics', Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pp. 311–318.
URL: *https://www.aclweb.org/anthology/P02-1040* [Online; Accessed: 24-April-2021]

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* pp. 2825–2830.
URL: *https://scikit-learn.org/stable/* [Online; Accessed: 24-April-2021]

Popel, M., Tomkova, M., Tomek, J., Kaiser, Ł., Uszkoreit, J., Bojar, O. & Žabokrtský, Z. (2020), 'Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals', *Nature Communications* **V**ol. 11(1), 4381. doi: 10.1038/s41467-020-18073-9.
URL: *https://doi.org/10.1038/s41467-020-18073-9* [Online; Accessed: 13-May-2021]

Python Software Foundation (2021), 'Python 3 documentation - built-in types'.
URL: *https://docs.python.org/3/library/stdtypes.html#str.casefold* [Online; Accessed: 24-March-2021]

Řehůřek, R. & Sojka, P. (2010), Software Framework for Topic Modelling with Large Corpora, *in* 'Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks', ELRA, Valletta, Malta, pp. 45–50.

Tiedemann, J. (2012), Parallel data, tools and interfaces in opus, *in* N. C. C. Chair), K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk & S. Piperidis, eds, 'Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)', European Language Resources Association (ELRA), Istanbul, Turkey.
URL: *https://www.aclweb.org/anthology/L12-1246/* [Online; Accessed: 24-April-2021]

Tiedemann, J. & Thottingal, S. (2020), OPUS-MT – building open translation services for the world, *in* 'Proceedings of the 22nd Annual Conference of the European Association for Machine Translation', European Association for

Machine Translation, Lisboa, Portugal, pp. 479–480.
URL: *https://www.aclweb.org/anthology/2020.eamt-1.61* [Online; Accessed: 24-April-2021]

Tomek, I. (1976), 'Two modifications of cnn', *IEEE Transactions on Systems, Man, and Cybernetics* **V**ol. SMC-6(11), 769–772. doi: doi:10.1109/TSMC.1976.4309452.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q. & Rush, A. M. (2020), Transformers: State-of-the-art natural language processing, *in* 'Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations', Association for Computational Linguistics, Online, pp. 38–45.
URL: *https://www.aclweb.org/anthology/2020.emnlp-demos.6* [Online; Accessed: 24-April-2021]

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. & Dean, J. (2016), 'Google's neural machine translation system: Bridging the gap between human and machine translation', *CoRR* .
URL: *http://arxiv.org/abs/1609.08144* [Online; Accessed: 24-April-2021]

Zhang, J., Zhao, Y., Saleh, M. & Liu, P. J. (2019), 'PEGASUS: pre-training with extracted gap-sentences for abstractive summarization', *CoRR* **V**ol. abs/1912.08777.
URL: *http://arxiv.org/abs/1912.08777* [Online; Accessed: 24-April-2021]

Šmerk, P. & Rychlý, P. (2009), Majka – rychlý morfologický analyzátor, Technical report, Masarykova univerzita.
URL: *http://nlp.fi.muni.cz/ma/* [Online; Accessed: 2-May-2021]

Appendix A

Acronyms

CNN	Condensed Nearest Neighbours.
EMEA	European Medicines Agency.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transport Protocol.
MT	Machine Translation.
NMT	Neural Machine Translation.
POS	Part Of Speech.
SGD	Stochastic Gradient Descent.
SMT	Statistical Machine Translation.
SVC	Support Vector Classifier.
URI	Uniform Resource Identifier.

Appendix B

Text origin search data sources

Robot: `https://en.wikipedia.org/w/index.php?title=Robot`
Bitcoin: `http://en.wikipedia.org/w/index.php?title=Bitcoin`
Ray tracing: `http://en.wikipedia.org/w/index.php?title=Ray%20tracing%20(graphics)`
Data Science skills: `https://towardsdatascience.com/how-to-make-more-money-as-a-data-scientist-bd6edd4fe460`
30 prediction models: `https://towardsdatascience.com/how-to-run-30-machine-learning-models-with-2-lines-of-code-d0f94a537e52`

Appendix C

Contents of enclosed SD card

```
README.MD .................... the file with SD card contents description
├─ data ......................................................... data directory
│  ├─ aligned ........................................... raw aligned data
│  └─ datasets ....................................... processed datasets
├─ files ....................................................... logs, files, etc..
├─ notebooks ............................... notebooks with experiments
├─ results .................................... pdf and csv files of results
├─ toolkit ........ frequently used funcionality in form of a python module
└─ thesis ...................................... LaTeX codes of the thesis
   └─ thesis.pdf ................................................... thesis
```