Università degli Studi di Milano
Facoltà di Scienze e Tecnologie
Master programme in Computer Science

UNSUPERVISED ERROR CORRECTION FOR OCR ITALIAN TEXT

Supervisor: Prof. Alfio FERRARA

Co-supervisor: Darya SHLYK

Thesis by:
Anton Shchedro
Matriculation: 989297

Academic Year 2022-2023

Contents

Chapter 1

Introduction

Grammatical Error Correction (GEC) is one of the NLP tasks and in itself a large field of research. Literature about GEC is divided into general analysis of this task, mostly made with English datasets and sometimes with multilingual datasets, and for some specific language or with texts form some specific field.

In general, Grammatical Error Correction also includes Grammatical Error Detection and sometimes Error Recognition [1]. GEC includes several types of error correction: misspellings, incorrect word order, and incorrect word usage (for example, incorrect verb tenses).

Most GEC techniques work with all types of errors, which makes working with only one particular type problematic because the models will "correct" other "errors" of unimportant error types when it's not necessary. For example, if we only work with misspellings, while other types of errors are considered redundant due to the fact that these types of errors are not present in the text, the model may consider correct text as an error and correct it.

This could be considered as critical issue in case when we try to detect and correct errors after OCR of scanned text, especially when style of speech is different from the common usage in that language, both by the passage of time, for example different generations, or by standards of government agencies [2] and that style has to be kept intact.

This paper implements [1] and evaluates several GEC methods, while also identifying and explaining errors and imperfections in the methods and the dataset used.

[1] https://github.com/Anton-Shchedro/GEC-Misspell-Italian

Chapter 2

State-of-the-art

Due to the fact that Grammatical Error Detection (GED) is a subtask of Grammatical Error Correction, there are fewer papers that concentrate only on GED, but are present in papers about GEC.

2.1 Grammatical Error Detection

GED could be done by checking word by word (or n-grams), or by checking the whole sentence at once.

For GED we can consider that True Positive (TP) results are those that contain errors and the detector has found them, False Positive (FP) results are words that are correct but marked as incorrect by the detector, False Negative (FN) results are words that contain errors but the detector does not mark them as containing errors.

2.1.1 N-grams

One of the basic techniques is a simple dictionary lookup: If a word isn't in the dictionary, it's probably misspelled.

One of the possible techniques proposed by Kernighan et al. is the use of a noisy channel model: the most probable correction c of a typo t is $c = argmax_c P(c)P(t|c)$, where $P(c)$ is a frequency of the word c in a corpus, and $P(t|c)$ can be computed as the inverse of the Damerau-Levenshtein edit distance between t and c [3]. This method successfully corrects 87% of typos.

However, this method is highly dependent on the corpus or dictionary, and the correct word may be marked as misspelled if the correct word does not exist in the dictionary. This can lead to False Positive results. Another issue is misspellings that transform one word into another existing word, for

example, 'cat' may be misspelled as 'cap', but for dictionary lookup both words are correct and we may get an False Negative result.

One possible way to improve the result is to use n consecutive words and a dictionary of n-grams. This helps to conserve some level of context for the word being examined.

When dealing with n-grams, a larger value of 'n' increases the likelihood of missing n-grams from the text in the dictionary, resulting in False Negative results. For example, in Mykowiecka and Markiniak's work, they used bigram language model that improved Kernighan et al's result up to 93.4%, but 57.6% of correct words were badly changed [4].

An additional problem with GED is name entities. Big portions of names, surnames, and company or organization names may not be present in standard dictionaries, causing GED to mark them as errors. An effective solution is to use NER tools prior to GED to ignore name entities during the search for errors[5].

2.1.2 POS tags

Some techniques preprocess sentences into Part-of-speech (POS) tags to search for errors using them. For example, Chodorow and Leacock [6] took adjacent part-of-speech tags and function words (determiners, prepositions, conjunctions, etc.) to search parts and "computes a mutual information measure to determine which sequences of part-of-speech tags and function words are unusually rare and therefore likely to be ungrammatical in English" by following formula for bigrams:

$$MI = log_2 \left(\frac{P(AB)}{P(A)XP(B)} \right) \tag{2.1}$$

where: "P(AB) is the probability of occurrence of the AB bigram, estimated from its frequency in the general corpus, and P(A) and P(B) are the probabilities of the first and second elements of the bigram".

Gamon [7] suggested using a Maximum Entropy Markov Model on POS to annotate parts of correct sentences and parts containing errors: Gamon modified the annotation from a NER task that contained tokens 'O' (outside of NE), 'B' (beginning of NE) and 'I' (inside of NE) to use only 'O' and 'I' tokens, resulting in only two states that can be easily transitioned from one state to another. The transmission of states is constrained by a set of features extracted from the text. There are 29 features computed from n-gram probabilities observed on and around the current token, grouped into 5 categories:

- Basic features - probability of token w and average n-gram probability of all n-grams in the sentence that contain w,
- Ratio features - ratio of the average x-gram probability of all x-grams containing w to the average y-gram probability of all y-grams containing w, for each combination where $x > y$, for example $x = 5$ and $y = 4$,
- Drop features - decrease or increase of the probability over the token w in n-gram,
- Entropy delta features - an alternative measure of changes in n-gram probability, *forward entropy* from w_i to w_n, where w_n is a last token in a sentence, and *backward entropy* from w_0 to w_i. Here, the entropy of an n-gram is defined as the language model probability of string $w_i \dots w_n$ divided by the number of tokens in that string,
- Miscellaneous - *minimum ratio to random*, *average ratio to random*, *overall ratio to random* (how the probability of the searched n-gram is different from the probability of the n-gram in random order) and *overlap to adjacent ratio* (the sum of the probabilities of the n-grams that include w_i, divided by the sum of the probabilities of the n-grams that are adjacent to w_i but do not include it).

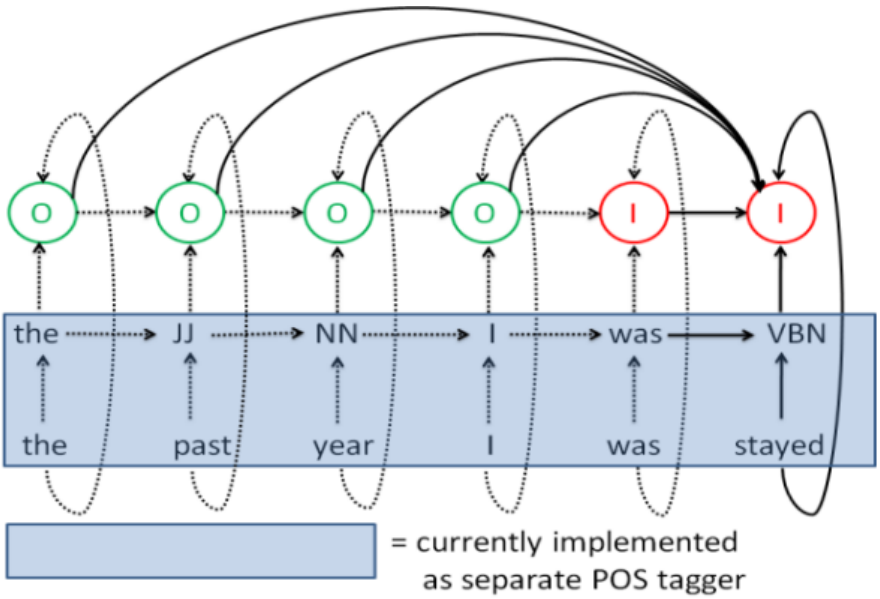

Figure 2.1: MEMM model for error detection[7]

2.1.3 Neural Networks

In 2016, Marek Rei and Helen Yannakoudakis [8] attempted to develop and analyse various neural network architectures for the GED. These included Convolutional, Bidirectional Recurrent, Bidirectional LSTM, and Deep versions of these architectures. They used a window to analyse only a sequence of words at a time by representing them as a vector $[w_1, ..., w_T]$ and the network extracted from this vector an array of hidden vector representations of each token in the context $[h_1, ..., h_T]$ to then pass through the softmax layer.

Bidirectional RNN and LSTM are types of RNN where information flows in both forward and backward directions. This enables the network to process information in the context of both preceding and succeeding tokens.

Evaluations made by Rei and Yannakoudakis of thous networks on First Certificate in English dataset [9] showed that the best precision was made by Bidirectional Recurrent (51.3), while the best recall and $F_{0.5}$ by Bidirectional LSTM (28.5 and 41.1 respectively), and after adding several additional training data $F_{0.5}$ score of Bidirectional LSTM increased up to 64.3. Test on CoNLL-14 shared task test dataset shown $F_{0.5}$ score of 44.0 [8].

$$P = \frac{TP}{TP + FP} \tag{2.2}$$

$$R = \frac{TP}{TP + FN} \tag{2.3}$$

$$F_\beta = (1 + \beta^2) \times \frac{P \times R}{(\beta^2 \times P) + R} \tag{2.4}$$

2.1.4 Transformer LM

One of the main focuses of research in the last decade for GED and GEC methods has been through the use of Transformers. This family of models consists of deep neural networks with an attention mechanism. Transformers do not have recurrent connections, which means they require less time for training and are better suited for parallelism. As a result, they are better for training on large datasets

The architecture of Transformers can be divided into two types of layers: encoder and decoder. The encoder retrieves information from a sequence of tokens, such as a sentence, while the decoder generates tokens based on previously inserted or generated tokens. Based on these two types of layers, all Transformer models are divided into three types:

- The encoder-only models are primarily used for text classification tasks. One of the most commonly used transformers is BERT and its modifications,
- The decoder only models are used for word generation, most commonly known from GPT models,
- The encoder-decoder models can be used, for example, for language translation tasks.

One of the transformers used for GED is EliCoDe by Colla et al. ([10], which is based on XLM-RoBERTa model. This model uses XLM-RoBERTa for feature extraction, a dropout layer to avoid overfitting, and a linear classifier to generate a 'correct' or 'incorrect' label for each token in a sequence. The authors used the BIO labelling schema, where B stands for 'beginning of the error unit', I stands for 'inside of the error unit', and O stands for 'outside of the error unit'. However, for simplicity, I was not used.

Colla et al. compared their model with several others by several metrics: precision, recall and $F_{0.5}$ score over MultiGED dataset[11] which contains several datasets with different languages. The EliCoDe method showed the best $F_{0.5}$ score for almost all the datasets, except for "English - REALEC".

One of the models compared with EliCoDe is NTNU-TRH, which shows approximately the best and sometimes better precision than EliCoDe. NTNU-TRH architecture [12] is a Bi-LSTM-CRF model (Bidirectional LSTM model with Conditional Random Field as classifier).

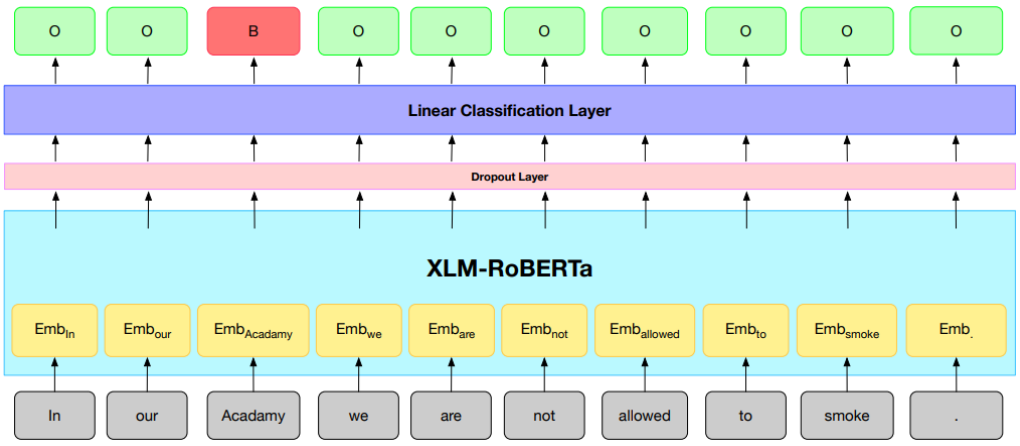

Figure 2.2: EliCoDe model for error detection[10]. Here the " incorrect" label is marked as B for "beginning of error" and O for "correct" token.

Conditional random field is a conditional model $p(Y|X)$ of two jointly distributed random variables X and Y. By definition [13] (X, Y) is a conditional random field if, when conditioned on X, the random variables Y_v obey the Markov property with respect to the graph: $p(Y_v|X, Y_w, w \neq v) = p(Y_v|X, Y_w, w \sim v)$, where $w \sim v$ means that w and v are neighbours in the graph $G = (V, E)$, such that $Y = (Y_v)_{v \in V}$, so that Y is indexed by the vertices of G.

System	Czech			English - FCE			English - REALEC		
	P	R	F0.5	P	R	F0.5	P	R	F0.5
NTNU-TRH	80.65	6.49	24.54	81.37	1.84	8.45	51.34	1.13	5.19
EliCoDe	82.29	50.61	73.14	73.64	50.34	67.40	44.32	40.73	43.55
EliCoDe$_{ALL}$	82.01	51.79	73.44	71.67	50.74	66.21	43.69	40.74	43.07

System	German			Italian			Swedish		
	P	R	F0.5	P	R	F0.5	P	R	F0.5
NTNU-TRH	83.56	15.58	44.61	93.38	19.84	53.62	80.12	5.09	20.31
EliCoDe	83.87	71.89	81.16	85.63	66.69	81.03	80.56	67.50	77.56
EliCoDe$_{ALL}$	84.78	73.75	82.32	86.67	67.96	82.15	81.80	66.34	78.16

Table 2.1: Results of experiments in the token classification task of the MultiGED2023 shared task. P stands for precision and R for recall. EliCoDe and EliCoDe$_{ALL}$ are two different experimental settings. EliCoDe was trained on the training set, while the development set was used to select the model. EliCoDe$_{ALL}$ was trained on both training and development sets.

2.2 Grammatical Error Correction

Some basic techniques have been made by different classification algorithms, such as decision trees [14] and Support Vector Machines[15]. However, most classifiers are only capable of fixing limited types of errors, for example only preposition (*a, an, the*), while errors such as noun number (singular or plural noun form) and verb form are not able to be determined and fixed by these models. Furthermore, the classifiers are only able to process one n-gram of fixed length at a time, rather than the whole sequence of words. This allows for parallel processing but may eliminate or reduce sentence context. Due to these limitations, classifiers are rarely used today, except as an adjunct to other techniques.

Some Grammatical Error Correction techniques based on statistics of the target sequence and the corpus have already been described in a section of

Grammatical Error Detection, such as the Noisy Channel model, and will not be described in this section.

2.2.1 RNN

Recurrent Neural Networks are designed to process the variable length input sequence. Like most neural networks for GEC task, RNN also use encoder-decoder architecture.

For example, in Zheng Yuan and Ted Briscoe's model [16], bidirectional RNN encoder reads and encodes the whole sequence into a vector, and attention-based decoder generates words with respect to the encoded vector and previously generated words.

Other types of RNN, such as Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRU), are also used for GEC. In their study, Xie et al. [17] used GRU in forward, backward and combined activation functions for the encoder and hidden activation function for the decoder, but unlike other models, this is a character-level sequence-to-sequence model.

2.2.2 CNN

Another method for GEC is the use of Convolutional Networks. These networks exploit the method of representing words as embedding vectors with positional embedding, which allows a sequence of words to be represented as a matrix and one-dimensional convolutional operations to be performed on the convolution kernel.

Convolutional layers nowadays can be seen in both encoders and decoders with attention mechanisms. For example, Chollampatt and Ng[18] used 7 convolutional layers in both encoder and decoder and had on the CoNLL-2014 test set a precision of 68.13, recall 23.45 and $F_{0.5}$ 49.33 on the basic version and a precision of 65.49, recall 33.14 and $F_{0.5}$ 54.79 with additional corpora, with re-scoring using edit operation and language model features and additional spell checking using statistical machine translation.

2.2.3 Transformers

Transformers are a powerful tool in modern GEC due to the amount of data used during training and the ability of the encoder to retrieve the meaning of a sentence. They can be used with different approaches:

- Translation Task - As part of Text2Text generation, the Translation task aims to translate sentences from one language to another. In

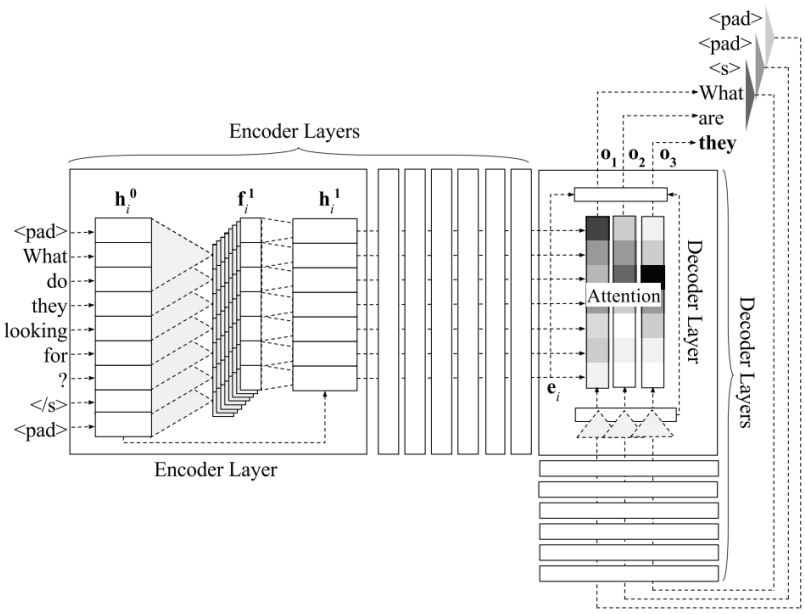

Figure 2.3: Architecture of Chollampatt and Ng[18] multilayer convolutional model with seven encoder and seven decoder layers.

the GEC, the aim is to translate an "incorrect" sentence into a "correct" one. For example, Kementchedjhieva and Søgaard [19] used the Language Translation Transformer as a round-trip translation for the intermediate result, which combined with the original sentence passed through the *mT5* text2text Transformer produces the correct sentence. Since this method requires multiple translations, the computational cost of this method is high.

Although it is possible to translate to the same language as the original text without an intermediate language using some Language Translation models, which reduces computational costs and can still be valid as a reference text.

- The Fill-Mask task - With the help of GED it is possible to mark which part of the sentence is "incorrect" and by replacing incorrect words as mask tokens generate "correct" words in place of "incorrect" ones. This can be done because the Fill-Mask model attempts to replace the mask with a word that is more probable given the context of the sentence.

- The Text Generation task - Text Generation models, such as the GPT family of models, are capable of generating some meaningful text, answering questions to some extent, and understanding text queried. This

creates the possibility to "ask" text generation models to correct source text.

For example, Loem et al.[20] examined the performance of GPT-3 for the GEC task and demonstrated that GPT achieves a GLEU score that is at least competitive, if not better, than some representatives of other types of transformers.

One of the drawbacks is that transformers is that they may fail to maintain the original word order or use synonyms for certain words. This is because the training dataset for these models may favour a particular version of a sentence or word.

Another issue is that some transformers may struggle to understand the meaning of certain words if they are corrupted and their meaning is isolated from the rest of the sentence (most likely in the case of name entities). This problem is not unique to transformers and can occur with other methods as well.

2.2.4 Generation of edit sequence

The generation of an edit sequence is a method that creates a sequence of edits to be applied to an 'incorrect' sentence, rather than generating a 'correct' sentence. The idea behind this method is based on the observation that a large part of the 'incorrect' sentence and the 'correct' sentence after GEC are the same. Therefore, generating correct tokens is computationally wasteful.

For instance, Stahlberg and Kumar [21] suggest generating edit operations on the source sentence, such as copying, deleting, or replacing it with one or more 'correct' tokens. Each edit operation is described by a tuple (t_n, p_n, r_n), where $t_n \in T$ and T is the tag vocabulary (e.g. SELF represents a copy operation of a token and SPELL represents a correction of a misspelled word), p_n is a position in the original sentence, r_n is a replacement token from the vocabulary with an additional DEL token for deleting the token.

The Stahlberg and Kumar model's architecture is based on Transformers, featuring one encoder and two decoder layers:

1. Decoder A generates information based on the previous iteration and Encoder. This information is then passed to the softmax function for tag prediction,

2. The Span End Position is predicted by the information from decoder A and the predicted tag using the attention weights over the encoder states as probabilities,

3. The information from Encoder, Decoder A, Tag, and Span End Position is embedded and fed to Decoder B,

4. Decoder B with the use of Softmax generates a replacement token.

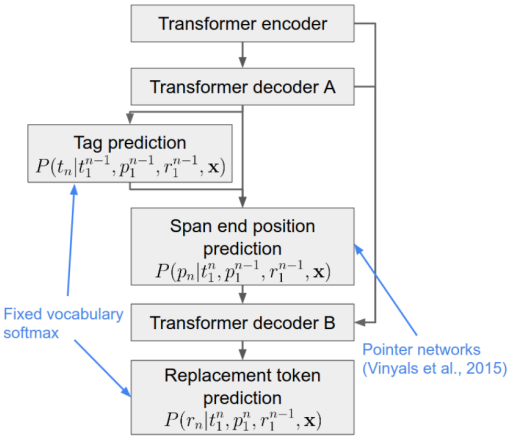

Figure 2.4: Architecture of Stahlberg and Kumar model [21].

Stahlberg and Kumar reported that their model for the GEC task achieved a precision of 69.9, a recall of 44.4 and a $F_{0.5}$ of 62.7 on CoNLL-14.

Chapter 3

Dataset

This paper utilises a dataset of transcripts from the Assemblies of the Chamber of Deputies of the Italian Republic from 1948 to the present day. Each record in the dataset includes various information, such as the date of the assembly, information about the deputy who made the speech recorded in that part of the transcript, the text of the transcript, etc., but for this paper only the stenographic text is important.

Most of the transcripts were created before the digitalization of texts became a common practice or was even available. For this reason, a large number of transcripts in this dataset were passed through OCR, resulting in some OCR-specific errors, such as:

- Misspelled characters - some characters may be misspelled as other characters. For example, character *m* can be misspelled as *n*, *rn*, *iii*, or others. Similarly, character *e* can be misspelled as *è* and vice versa.
- Missing character
- Extra characters - due to imperfect printing or scanning processes, defects may exist in proximity to the text. These defects may be seen by OCR as a part of the text and cause misspelled or extra characters. Most of the extra characters are non-alphanumeric, such as comma (,), apostrophe (’), or full stop (.) symbols.

  Additionally, a very common error is the presence of extra spaces within words, resulting in the transformation of one word into two or more words. In the worst case scenario, a word may be split into individual letters.
- Wrong line sequence - Some OCR processed documents may have lines recognised in the wrong order. Unfortunately, due to the structure of

the dataset, where records contain only text as a whole, it is impossible to correct this type of error in one of the preprocessing steps.

Besides the errors described above, there are also errors that occur during the retrieval of data from post-OCR documents:

- Transcripts may contain page transitions in the middle of a sentence. During transitions, it is not uncommon for headers to appear in the text. These headers may contain information about the document in general, but they are unrelated to the text.
- Some records may unexpectedly end in the middle of a sentence.

In both cases, error correction, even when done manually, can be difficult and may even be impossible.

Transcripts that are immediately made in digital format are not subject to OCR errors, but may still contain errors from the information retrieval process.

For evaluation purposes, a small subset of texts was randomly selected and corrected for OCR errors. The corrections were not made by a professional editor, so some errors may still be present, but the total number of errors is lower.

This subset contains both OCR processed transcripts and digital transcripts. Each record in this subset includes both the corrected text and the original text. In the case of digital transcripts, the corrected text and the original text are the same. This is because, as mentioned previously, digital transcripts do not contain OCR errors.

Chapter 4

Grammatical Error Detection

Several methods for Grammatical Error Detection were attempted in this work, which are described in the State of the Art. However, due to the specific nature of the errors being targeted (i.e. spelling errors), some of the methods reported in the State-of-the-Art were not selected.

Some of the following methods require a corpus of Italian texts.

4.1 Corpus

The PAISA corpus [22] was used, which is a corpus of freely available and freely distributable web texts. It consists of about 380,000 documents from about 1,000 different websites. Approximately 265,600 documents are from Wikipedia Foundation projects, and about 65,000 documents are from blogs.

As Wikipedia sites are reviewed by the community, the chances of grammatical errors may be lower than in blogs. To ensure the highest quality, only documents containing 'it.wiki' in their URL were included in the sub-corpus.

Multiple versions of this sub-corpus were created due to the use of different tokenizers and preprocessing methods.

4.2 Detection by probability

The first method utilises the approach similar to that proposed by Kernighan et al. This involves using the formula $argmax_c P(c)P(t|c)$, where $P(c)$ represents absolute frequency and $P(t|c)$ represents Levenshtein edit distance.

The use of 2-grams of words was preferred because, even though some words containing errors may be common in the corpus, 2-grams of words with errors have a lower frequency. A Bag-of-2-grams was created from a corpus, containing the absolute frequency of each 2-gram in the corpus.

Only around 10% of Bag-of-2-grams was used because big part of it contain 2-grams with low frequencies. The ordered Bag-of-2-grams by frequency yields the following statistics:

- The absolute frequency of the median 2-gram is 1, which is too low to be considered acceptable given the possibility of errors in the corpus.
- The absolute frequency of 2-grams in the top 25% is equal to 2.
- The absolute frequency of 2-grams in the top 10% is equal to 7.
- The absolute frequency of 2-grams in the top 1% is equal to 100.
- The highest frequency 2-gram has an absolute frequency of 1,450,528.

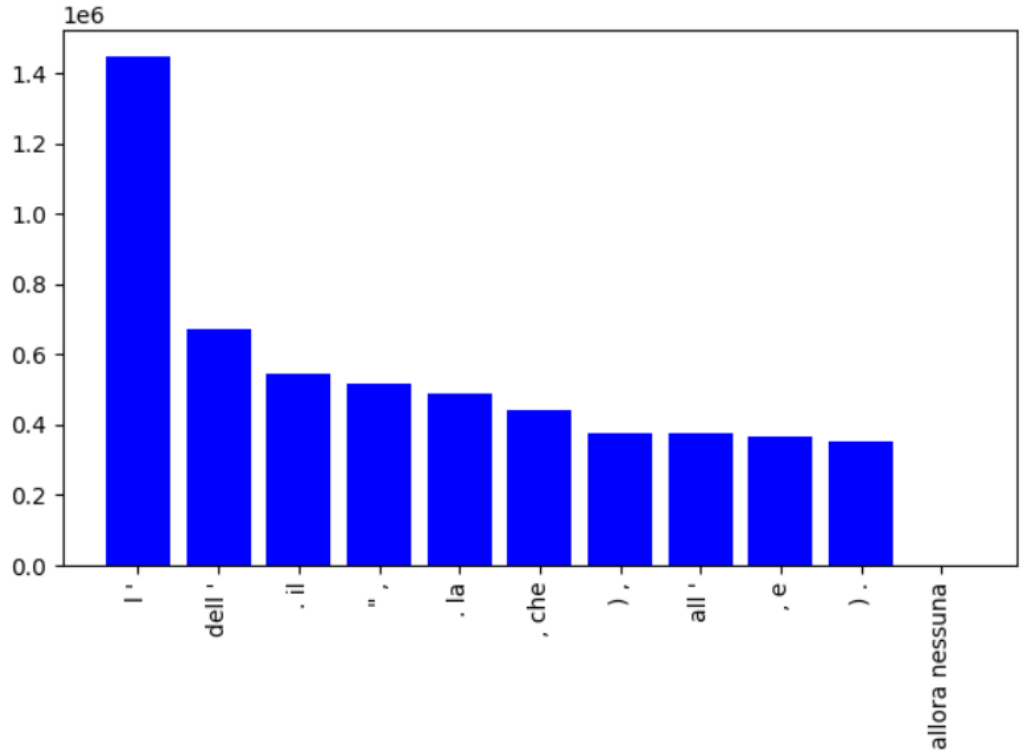

Figure 4.1: Absolute frequencies of 2-grams at the top 10 and one in the top 10%

Based on the statistics reported above, it may be reasonable to slice the Bag-of-2-grams up to 10%. This will allow us to remove 2-grams that contain errors and have low frequencies, while retaining those with high frequencies.

Some drawbacks in the approach:

1. Some common errors may be included in a sample that increases the False Negative results, as GED will show that 2-grams is correct, but in reality is not.

2. Some correct 2-grams that are not commonly used may be removed from the slice. This can increase the number of False Positive results when GED detects an error in the 2-gram, but it is actually correct.

3. The difference between the top one 2-gram and 2-gram in the top 10% is more than 200,000 times greater.

Due to the reasons outlined in the last point and in a plot **??**, using absolute frequency alone is useless, as the top 2-gram will almost always be the best candidate.

One possible solution is to use a logarithmic function on the probability: $argmax_c\ log(P(c))P(t|c)$.

Similar to a previous attempt, a slice was made at the same point and the following statistics were obtained:

- The frequency of 2-grams in the top 25% is equal to 0.69315,
- The frequency of 2-grams in the top 10% is equal to 1.94591,
- The frequency of 2-grams in the top 1% is equal to 4.60517,
- The highest frequency 2-gram has an frequency of 14.18744.

As shown above, the difference between top 1 and top 10% in about 7 times. This number is very high, and as result of this difference top 1 2-gram may be considered as best candidate for each 2-gram with Levenshtein edit distance less or equal to 7, that is for at least 242889 2-grams with size up to 7 characters.

As demonstrated previously, the difference between the top-ranked 2-gram and the 2-gram within the top 10% is about seven times. This high number indicates that the top 2-gram with a Levenshtein edit distance of 7 or less may be considered the best candidate for at least 242889 2-grams with a size of up to 7 characters.

Another possible variation of the previous method is to increase the value of n in n-grams. In the case of 3-grams, the total number of n-grams present in the corpus is increased as well as the number of low-frequency n-grams.

The following statistics show the frequencies of ordered Bag-of-3-grams:

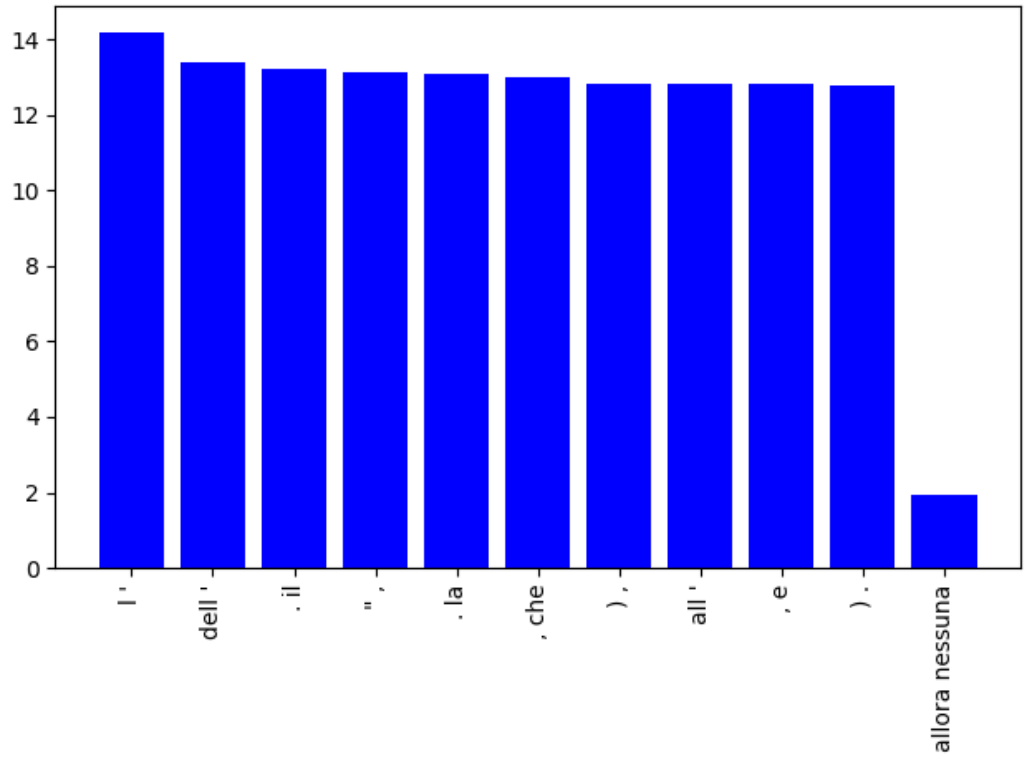

Figure 4.2: Absolute frequencies of 2-grams at the top 10 and one in the top 10% in logarithmic scale

- The absolute frequency of 3-grams in the top 10% is equal to 3.
- The absolute frequency of 3-grams in the top 5% is equal to 5.
- The absolute frequency of 3-grams in the top 2.5% is equal to 10.
- The highest frequency 3-gram has an absolute frequency of 208,167.

Due to the significant difference between the top 3-gram and the top 2.5% of 3-gram, which can be used as a slicing point with a dimension similar to 10% in 2-gram, the use of 3-grams may also be considered ineffective. Additionally, the logarithmic version of bag-of-3-grams has similar statistics to that of 2-grams, with the top frequency being 12.24610 and the top 2.5% (as the end of the slice) being 2.30259, with a difference of more than 5 times.

For these reasons, methods based on probabilities of 2-grams and 3-grams were considered ineffective.

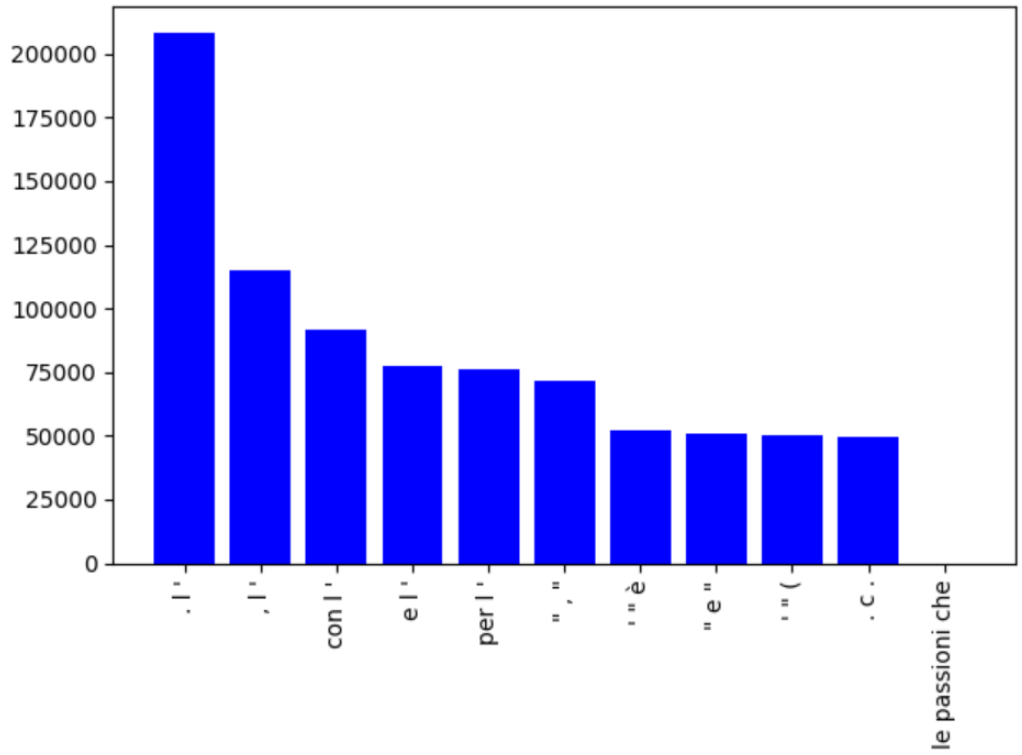

Figure 4.3: Absolute frequencies of 3-grams at the top 10 and one in the top 10%

4.3 Search in Bag-of-n-gram

One possible approach to use a slice of the Bag-of-2-grams for error detection is to simply check if there is a 2-gram in that slice: if there is a 2-gram in a slice then, assuming that slice contains no or a small number of errors, while most errors remain outside the slice, that 2-gram is most likely correct.

The search for 2-grams in the Bag-of-2-grams is straightforward:

1. The sentence is split into a list of tokens by the tokeniser. In this section of the paper, the tokenizer used splits the sentence by words and non-alphanumeric symbols.

2. Pair the items in the list after splitting. For example, if the list is ['a','b','c','d'], the pairs would be ['a b', 'b c', 'c d']

3. Search each pair in the slice of Bag-of-2-gram. The result of this search is a list of booleans with a length equal to the length of the list of

pairs. A True value is assigned if the 2-gram is found, and it can be considered error-free.

This implementation has one problem: the tokenizers used in this interpretation create 2-grams out of two words, while detection needs to be done for each word individually. To solve this problem, the following action was taken before returning the result:

4. Add padding consisting of 'True' boolean values to the beginning of the list of booleans (point 3) with a length of n-1 for the n-gram. For a 2-grams, add one 'True' value to the beginning of the list.

5. For each element $bool_i$ in a list of booleans up to $length(list) - (n-1)$ element make logical OR of set from $bool_i$ up to $bool_{i+(n-1)}$ where n from n-gram. For example, for a 2-grams, take the logical OR of $bool_i$ and $bool_{i+1}$.

The resulting list should have the same length as the number of tokens in the sentence. For example, if there is one token for each word, the list should be the same size as the number of words in the sentence. This list can be used as a result of GED, where 'True' indicates a correct word and 'False' indicates a word containing an error.

4.4 Various modifications

Multiple tokenizers and preprocessings were tested:

- Tokenizer by split by regex $r' \setminus s|([\hat{}\setminus w])'$,
- Tokenizer by *spaCy* [23] Python library,

The evaluation (which will be described later in this section) comparing these two tokenizers shows that tokenizing by regex produces a better $F_{0.5}$ score.

Additionally, some preprocessing techniques were used:

- Regex tokenizer and lemmatizer by *spaCy*,
- Regex tokenizer and Name Entity Recognition by *spaCy*,

The Lemmatizer was used to reduce words to their lemma, which helps to identify misspelled words that cannot be reduced. For example, the word

'ritornano' has the lemma *'ritornare'*, while the misspelled version of the same word, *'iitornano'*, has the lemma *'iitornano'*, which is the same.

As mentioned in the State-of-the-Art section, Name Entity Recognition (NER) can also improve Grammatical Error Detection. For the Italian language, the SpaCy version of NER was used with one modification: in the Italian version, there is a small number of NER labels, one of which is 'Miscellaneous entity'. During some trials before evaluation, it was found that errors were often labelled by NER as 'Miscellaneous entity'. Therefore, all words labelled as 'Miscellaneous entity' were not processed as Name Entities.

In both cases, the corpus was passed through appropriate preprocessing techniques and sequentially through the tokenizer to obtain Bag-of-2-grams: in the case of the lemmatizer, every word in the corpus was replaced by its lemma, while in the case of the NER, every word that was recognised as a Named Entity, with the exception of Miscellaneous entities, was replaced by its label.

During sentence processing by GED, the sentence is first passed through preprocessing then the process is the same as described in the Search in Bag-of-n-gram section, with corresponding tokenizer and Bag-of-n-gram preprocessing technique.

4.5 Transformer

As stated in Section 2.1.4 of the State-of-the-Art, transformers are currently one of the most powerful tools for Natural Language Processing. For Grammatical Error Detection, the EliCoDe [10] transformer was chosen due to its high performance and support for the Italian language. The structure of this transformer was also described in Section 2.1.4.

To use EliCoDe correctly, all texts must be tokenized and kept in TSV format. For each token, it is necessary to append the string *'O O O'*. The output of EliCoDe processing is in TSV format: a token and the string *'O'* for correct tokens or *'B-Error'* for tokens with errors.

To make it easier to use and to speed up the computation, a small amount of code has been added to bypass the need to save the input text to a file and read the output of the transformer from another file.

Furthermore, as some texts exceed 1000 tokens in length, they are split into sentences and only a small number of sentences are passed to the transformer at the same time.

Table 4.1: Evaluation of Grammatical Error Detection methods described above

Method	Precision	Recall	$F_{0.5}$
2-gram regex	0.21444	0.32522	0.20409
2-gram spaCy	0.19746	0.26213	0.19092
2-gram lemma regex	0.25300	0.12161	0.17785
2-gram lemma spaCy	0.12140	0.39453	0.13223
2-gram NER spaCy	0.19521	0.34267	0.19731
EliCoDe regex	0.31689	**0.42091**	0.31135
EliCoDe spaCy	**0.36686**	0.38583	**0.35042**

4.6 Evaluation of Grammatical Error Detection

The corrected text and original text from a small subset of manually corrected transcripts, as described in section 3, were compared using the tokenizers described in section 4.4 to identify differences. The difference between two texts is stored in the form of a list of booleans, where a "True" value is for the same token in both texts, while a "False" value is for different tokens in the texts. This difference was then used as the ground truth for evaluating Grammatical Error Detection.

Multiple rule sets are used to make this comparison. As a result, some small parts of the difference between texts may be incorrect. Additionally, there are errors in the original text that were not corrected (see section 3), which furthermore reduce the $F_{0.5}$ score.

Both lists, the difference list and the error detection list, mark the wrong word as a "False" value, for the following evaluation a True Positive is considered, if both lists have the value "False" at a moment i. If the difference shows no error (has a value of "True"), but the GED shows an error (has a value of "False"), this error is considered as extra and False Positive. If the difference show error (have a value of "False"), but the GED does not (have a value of "True"), this error is considered as missed and False Negative.

For each text, the Precision, Recall and $F_{0.5}$ scores were calculated separately, and the mean of these metrics over the subset is shown below.

When comparing simple search in Bag-of-2-gram, the regex tokenizer shows better performance than the tokenizer by spaCy. Replacing words

with lemmas does not improve performance. However, using Named Entity Recognition has a slightly better result than simple search in Bag-of-2-gram with spaCy tokenizer.

In contrast, the EliCoDe transformer shows significantly better results than the search methods in Bag-of-2-gram. When using the spaCy tokenizer, EliCoDe achieves better Precision and $F_{0.5}$ scores, while using the regex tokenizer achieves better Recall.

For the Grammatical Error Correction method based on the Fill-Mask task transformer, both versions of Grammatical Error Detection by EliCoDe will be used. EliCoDe with spaCy tokenizer will be used in the GEC evaluation due to its best $F_{0.5}$ score. Additionally, EliCoDe with regex tokenizer will also be used, as the Fill-Mask task transformer can potentially negate some False Positive errors.

Chapter 5

Grammatical Error Correction

In this paper two methods of grammatical error correction based on transformers have been implemented. One method uses Grammatical Error Detection, while the other does not.

5.1 Fill-Mask

Because Grammatical Error Detection can indicate the location of errors, it is possible to replace each error with a mask and run a Fill-mask transformer to retrieve a number of candidates for error correction. The most similar candidate can then be selected for correction.

For this implementation, a transformer for the Fill-mask task is based on the XLM-RoBERTa model pre-trained by Facebook [24].

The general structure of implementations is:

1. For each text, separate it into sentences and group them in sets of five.
2. For each set of sentences, find the errors by using EliCoDe (refer to section 4.5) and the corresponding list of words.
3. The list of words and errors resulting from the process of set of sentences is passed on to an Error Correction. The list of errors is represented by a list of Boolean values indicating whether a word is correct or not:
   (a) Iterate through the list of errors:
   If Error Detection shows that word is correct, add that word to the list of checked words. If Error Detection indicates that the word contains an error, add the index of the error to a list of tokens to be corrected (error group). If next word is incorrect repeat this part until the end of the continuous error sequence.

(b) Concatenate the list of checked words with the list of words from the Grammatical Error Detection, starting from the currently tested word. This will create a list of words where all words before an error group have already been processed, and words after the errors, including the last word from the error group.

(c) Replace the element of the concatenated list immediately following the checked word, which is an erroneous word, with a mask. Then, convert the resulting list into a string.

(d) Using the Fill-mask transformer, pass the created string with a mask to generate n candidates.

(e) Calculate the Levenshtein edit distance between each candidate and the string created by a group of incorrect words ($error\_string$).

(f) The best candidate is chosen as the one with the lowest distance, and if the Levenshtein distance is lower than the $length_{error\_string} * \alpha$ where α is a parameter between 0 and 1, the best candidate is added to a list of checked words. Otherwise, the error is retained.

Any incorrect word with a length lower than δ is kept in any case. This is done because a small list of candidates can have the same distance and almost any candidate with a small number of characters can be selected. Additionally, many words with small lengths have a difference of one character, such as prepositions.

4. The method returns the resulting list of checked words as a list of tokens of correct sentences, and appends it to the result of the previous step. This list is returned at the end of the entire process as the final result. By concatenating this list of words, we obtain the final sentence as a string.

The creation of a single mask for a group is based on the assumption that the probability of two consecutive errors in two different words is lower than if one word is split into a multiple of words.

The decision to retain errors is based on the fact that the number of candidates for the same token can be significantly larger than the number of selected candidates. In rare cases, this number may exceed a reasonable number of selected candidates.

Several modifications of this method were also tried:

- Create a mask for each error element within an error group. Then, compare a candidate for each mask with the respective incorrect word. This method can deal with the case where an assumption of probability

described above is incorrect, but this method will generate extra words in the case where the original word is split into several words.

- Select the best candidate not only by distance, but using a formula $Candidate\_Score * \beta^{distance}$, where β is a parameter between 0 and 1.

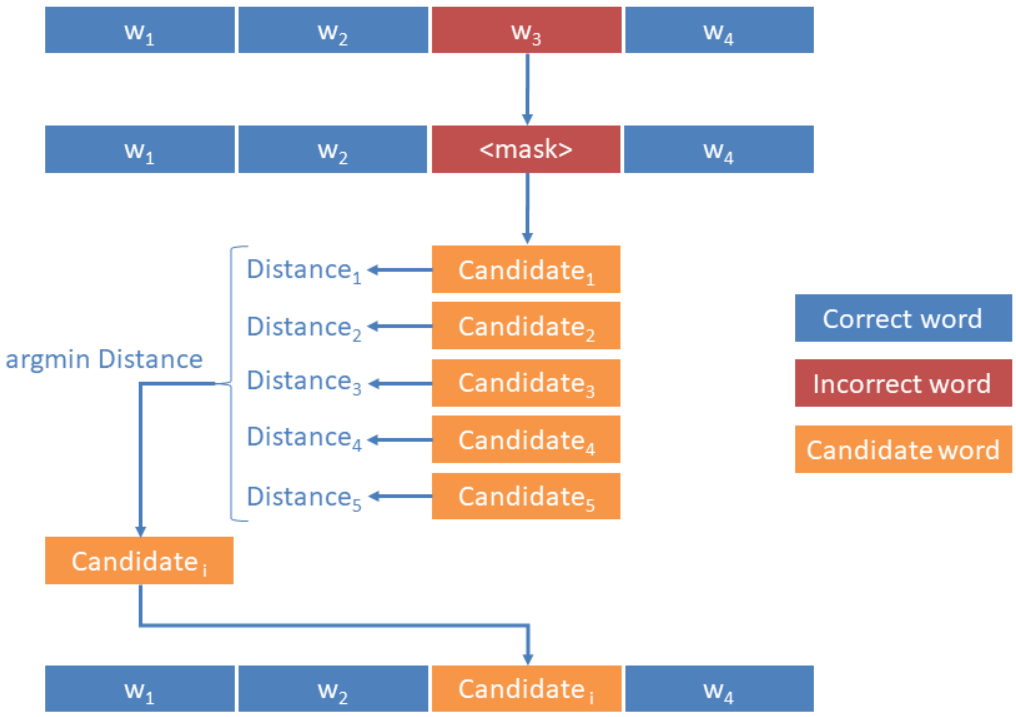

Figure 5.1: Abstract description of GEC by Fill-Mask transformer

To find the optimal combinations of parameters, these methods were run on a small portion of the subset of texts, which was approximately one-third the size of the manually corrected subset. The score for each text was calculated as the number of errors from GEC divided by the number of errors in the original text. This score can vary from 0 (when the GEC resulting text is the same as the ground truth) to greater than 1 (when the GEC resulting text has more errors than the original text). If the original text contains no errors and the resulting GEC text matches the ground truth, the score is set to 0. If the resulting GEC text does not match the ground truth, the score is set to 1. The evaluation is based on the mean value of the subset, and the lowest score is considered the best score.

- With spaCy tokenizer:

    1. The unmodified method (*FM spaCy V1*) achieved a score of 0.707839 for an α parameter of 0.3 and 0.708525 for an α parameter of 0.5. Varying δ, the minimum length of characters in an error substring to be kept, from 2 to 6 had no impact on this score. When δ was set to 7, the score increased slightly but not significantly. The difference in score between α values of 0.3 and 0.5 was minimal and can be considered negligible.
    2. The method that recalculates the best candidate using $\beta^{distance}$ (*FM spaCy V2*) achieved the best score of 0.733736 with α set to 0.5 and β set to 0.1. δ has been set to 4 due to little or no variation in the average score in the previous method.
    3. The method that uses a mask for each element in a error group and incorporates $\beta^{distance}$ (*FM spaCy V3*) has the worst result, with all scores higher than 1.0. The best score achieved by this method is 1.509540, indicating that it generates more errors than there were originally.

- With regex tokenizer:

    1. Similarly to the first method, but using the regex tokenizer (*FM regex V1*), this method achieves the best mean score of 0.937059 for α values between 0.3-0.4. This score is significantly higher than the method that uses the spaCy tokenizer. The variation of scores with changes in δ, as with other methods, is minimal.
    2. Similar to the second method, but using the regex tokenizer (*FM regex V2*). It achieved the best average score of 0.789259 for α equal to 0.6 and β in the range 0.1-0.4.

Plots representing the evaluation of all these methods are reported in the Appendix A.

The displayed numbers indicate that setting a minimum length for the error substring to be retained is ineffective. Instead, the parameters responsible for checking the Levenshtein distance between the best candidate and the incorrect word (α) or recalculating the score with Levenshtein distance (β) are significant.

Fixing errors one by one instead of as a group has proven to be ineffective and can result in more errors. This may be because some errors in a group are actually part of one word that has been divided by an unexpected space.

The Fill-Mask may try to replace the divided word with multiple words, causing further errors.

The selection of a tokenizer is crucial. The results show that the regex tokenizer performs worse than the spaCy tokenizer. This could be due to the lower precision of the Grammatical Error Detection method by EliCoDe with regex tokenizer than with spaCy tokenizer.

5.2 Translation

During the encoding phase, transformers can understand the context of the text to a certain extent. In theory, if an error does not significantly alter the context or make it unreadable, it is possible to rewrite the sentence without the error. This assumption was the basis for choosing a transformer for the translation task.

The selected model is the NLLB-200 distilled 600M variant by Facebook [25]. This model allows to translate from Italian to Italian without any intermediate language. However, it is important to note that the tokenizer used in this transformer does not tokenize each word as a separate token, but may create multiple tokens from a single word.

One issue with using this transformer only to correct misspellings is that it attempts to correct the less common sentence structure, which is not necessary. The main focus of this paper is to correct misspellings without changing the sentence structure.

To solve this issue, a Levenshtein distance calculation was added to the process of recalculating candidates' scores, and some additional modifications were made to adapt to this recalculation.

This model has the following structure:

1. Because this model runs on a Beam Search algorithm, at a certain point i, the model has n previously generated sequences of tokens.

2. For every generated sequence, predict K candidate tokens to append to a previously generated sequence. At this point, the model has a total of nK candidate sequences, which includes both previously generated sequences and predicted tokens.

3. For each candidate sequence replace tokens from position 0 up to i from original text with candidate sequence.

4. Calculate the Levenshtein edit distance for each resulting sequence and then recalculate the score for the same sequence. A formula:

$$Sequence\_score/(\beta^{distance})$$

is applied to obtain the final score of the candidate. The weight parameter, β, ranges between 0 and 1 and is used to balance the impact of distance on the score. If β is equal to 1, the Levenshtein edit distance has no effect. The formula includes division because the model returns a score in negative float numbers, with the best candidate having the highest score.

5. The model selects the n best candidate sequences based on the resulting score, and the process is repeated until the model predicts the end of the sequence.

One of the main drawbacks of this model is that it generates every token in a sentence, unlike the Fill-Mask transformer which only generates masked tokens. As a result, the Translation tokenizer takes significantly more time to process a sentence.

Furthermore, this model requires a large amount of allocated memory and for some texts this amount may be greater than available. For instance, while conducting an optimal parameter search on a computer with the following specifications: NVIDIA GeForce GTX 1060 6 GB GPU and 32 GB RAM, some sentences produced an error related to memory allocation (*torch.cuda.OutOfMemoryError*). This could be due to running a model inside another model to recalculate sentence scores, which exceeded the GPU's memory limit.

To minimise memory problems, the text was divided into sentences and the transformer processes one sentence at a time. However, this can result in the loss of information between sentences. Another possible solution is to use a model with fewer parameters, but this may lead to worse results.

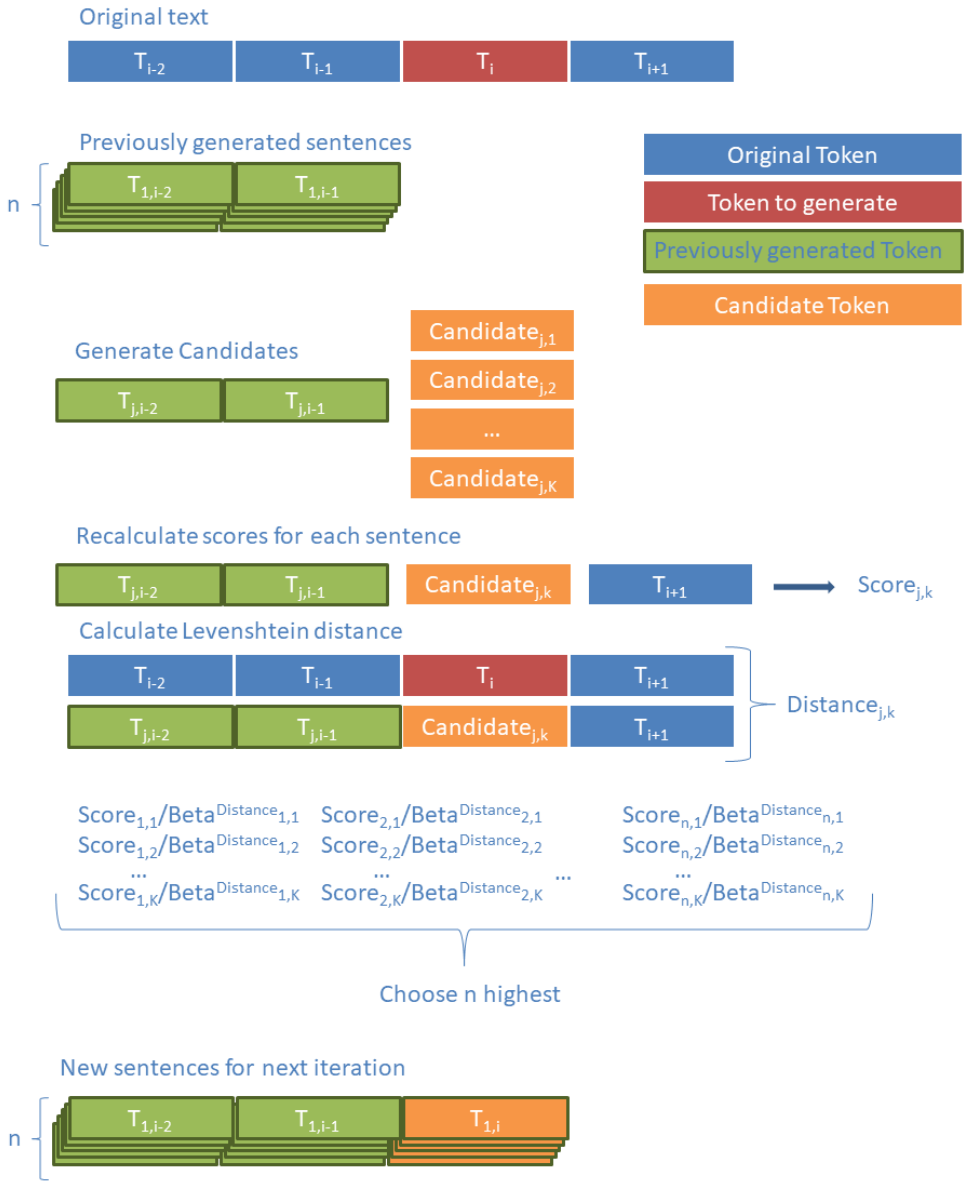

Figure 5.2: Abstract description of GEC by Translation transformer

Similar to the Fill-Mask method, the model was run to find the optimal parameter β over a same small portion of the subset of texts. The score for each text was calculated as the number of errors from GEC divided by the number of errors in the original text. For this method, β was tested in the range between 0.6 and 0.9. The test shows that a β value of 0.8 yields the best average score for this range of β with a result of 5.367794.

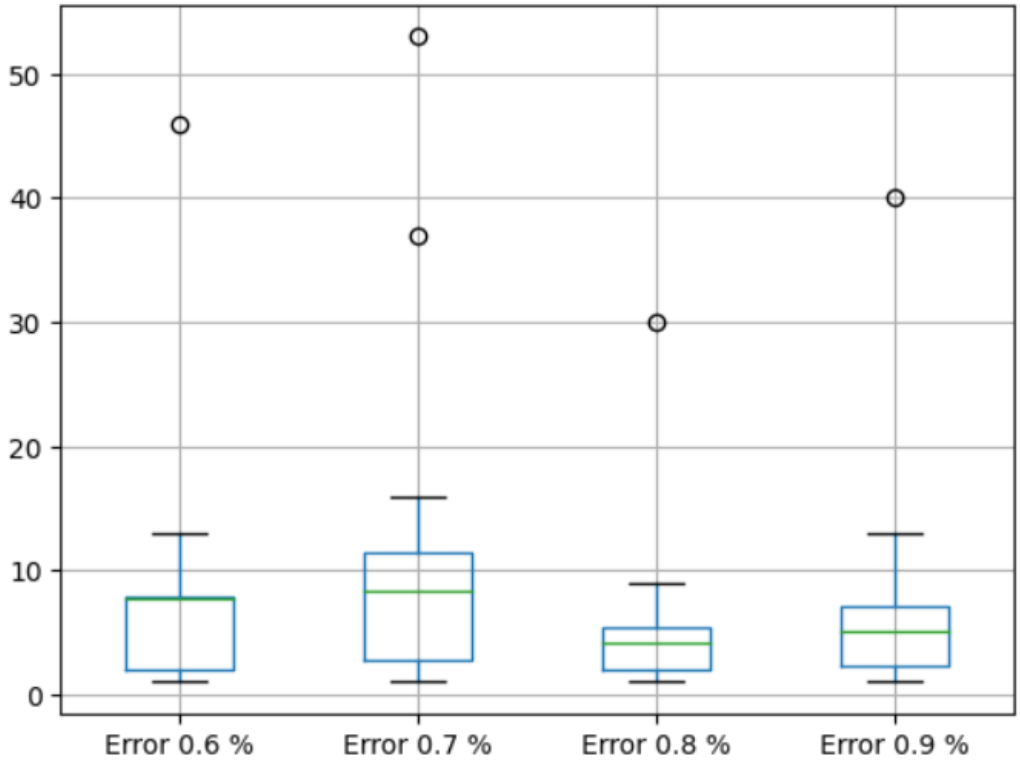

Figure 5.3: Boxplot of scores of GEC by translation

Chapter 6

Evaluation

For the final evaluation of the few best methods described in the previous section, a manually checked part of the subset not used in the parameter search was used. A total of 32 texts were used as a test set, of which 9 did not contain errors. On average, the percentage of errors in the text is 9.13% of tokens, and in some texts, it is as high as 49.1%.

The evaluation was performed using the same method as for the parameter search in the Grammatical Error Correction part, i.e. the number of different tokens between the text after Error Correction (later *GEC text*) and the Ground True text, divided by the number of different tokens between the original text and the Ground True text.

The resulting values are equal to 0 if the text is the same as the Ground True text, and greater otherwise. If the resulting value is equal to 1, the number of errors between the *GEC text* and the Ground True text, and between the Original text and the Ground Truth text, is the same.

If the original text and the Ground True text are the same, because of the division by zero problem, if the number of different tokens between *GEC text* and the Ground True text is 0, the score is set to 0, otherwise it is set to 1.

The chosen methods, which had the best scores during parameter search, are *FM spaCy V1* with an α value of 0.3 and 0.5, *FM spaCy V2* with an α value of 0.5 and a β value of 0.1, *FM regex V2* with an α value of 0.6 and a β value of 0.1, and Translation with a β parameter of 0.8.

As demonstrated above, *FM spaCy V1* can reduce the number of errors by about 33% on average, but it still produces errors. In some cases, it does not fix errors at all or even produces more errors than beforehand. This occurs in approximately one-third of cases. The same issue occurs with other Fill-Mask methods reported in the table above.

As noted in the evaluation section of Grammatical Error Detection 4.6,

Model name	Mean score	Min score	Max score
FM spaCy V1 α 0.3	**0.657092**	0.0	1.166667
FM spaCy V1 α 0.5	0.665649	0.0	1.333333
FM spaCy V2	0.716529	0.0	2.0
FM regex V2	0.86094	0.0	1.25
Translation	4.837326	1.0	49.250000

Table 6.1: Caption

the accuracy of the evaluation on this dataset is compromised by:

- Manual error correction was performed on the Ground Truth text. However, some errors may have been missed and left uncorrected.
- The structure of some retrieved texts is compromised by page headers, unexpected sentence endings in the middle of the sentence, and mixed rows.

These problems may cause additional errors to be generated and make it difficult to compare the text accurately.

Chapter 7

Error description

The errors in generation using this methods can be explained by several assumptions:

- Errors may occur if a word's meaning is isolated from the rest of the sentence without any supporting words. For example, in a sentence:

  "MORINI, Segretario, legge: « Per gli interventi di ..."

  the word *"Segretario"* (English: "Secretary") is written as *"Segretmio"* in the original text, and there is no significant connection between its meaning and other words that could suggest its correct meaning in the event of an error.

  As a consequence, transformers can propose a large number of candidates, all of which are correct from a grammatical point of view, but the candidates with the best scores by the transformer may not be words with the meaning closest to the original, but are more frequent words in the training set of that transformer.

  The same problem occurs with Named Entities. Translation transformers may be more susceptible to issues with Named Entities if the encoder component is unable to recognize a word as a Named Entity. In the case of Fill-Mask, Grammatical Error Detection can reduce this vulnerability to some extent.

  Synonyms and words of the same category can also be vulnerable to this issue. For example, in the text:

  "I like to eat cupcakes"

  the word *"cupcakes"* may have many possible candidates in its place. However, the word *"eat"* narrows down the options to the category of

food. Nevertheless, the number of words in this category may still be larger than a reasonable number of candidates to generate.

Issues with synonyms are particularly significant for Fill-Mask transformers, as replacing a word with a mask completely removes its meaning. The translation transformer may sometimes overlook this problem due to the possibility of using an original word as a reference.

The Levenshtein edit distance can partially solve this problem, but only if the correct word is included in the list of generated candidates.

- If one word is mistakenly replaced with another valid word, it may be difficult for Grammatical Error Detection to identify the error. Sometimes replacements can be detected because they can make a sentence grammatically incorrect.

  When a sentence with an error is grammatically correct, both the Grammatical Error Detection technique and the Grammatical Error Correction technique without Error Detection will miss the error. For example when:

  "I like this black cap."

  is replaced by:

  "I like this black cat."

  both sentences are grammatically correct and there are no errors that can be spotted.

  This error can be avoided if information about the incorrect word is present in nearby sentences. However, the translation Transformer receives one sentence at a time, and for memory consumption reasons, EliCoDe Grammatical Error Detection and Fill-Mast transformers pass chunks of a few sentences at a time. It may happen that two sentences, one with the incorrect word and the other describing the incorrect word, could end up in different chunks.

- The separation of a word into several words with unexpected spacing can be a big problem because it is sometimes difficult to proceed with this kind of error. It can be particularly challenging to automatically identify the correct group of incorrect words that form a single correct word when there are other incorrect words nearby that are not part of the group.

Chapter 8

Conclusion

This paper explores several methods for finding and correcting misspelled words in post-OCR texts. It has been demonstrated that using Grammatical Error Detection by EliCoDe [10] in combination with Fill-Mask Transformer for Grammatical Error Correction, specifically using XLM-RoBERTa [24], both implemented with spaCy [23], can effectively reduce errors introduced by OCR text recognition systems, particularly when the original file is unavailable for error correction.

Optimal parameters were found for the proposed method. However, it should be noted that due to the imperfections of EliCoDe and Transformers, the results may not be optimal and should be treated with caution.

This paper also covers some of the common errors that can be found in post-OCR text and some of the problems that can occur when processing them.

Due to the lack of specialised Grammatical Error Correction methods for correcting misspellings in Italian text, this work may be considered useful and can be used as a pre-processing step when working with datasets of OCR processed texts.

Appendix A

Fill-Mask plots

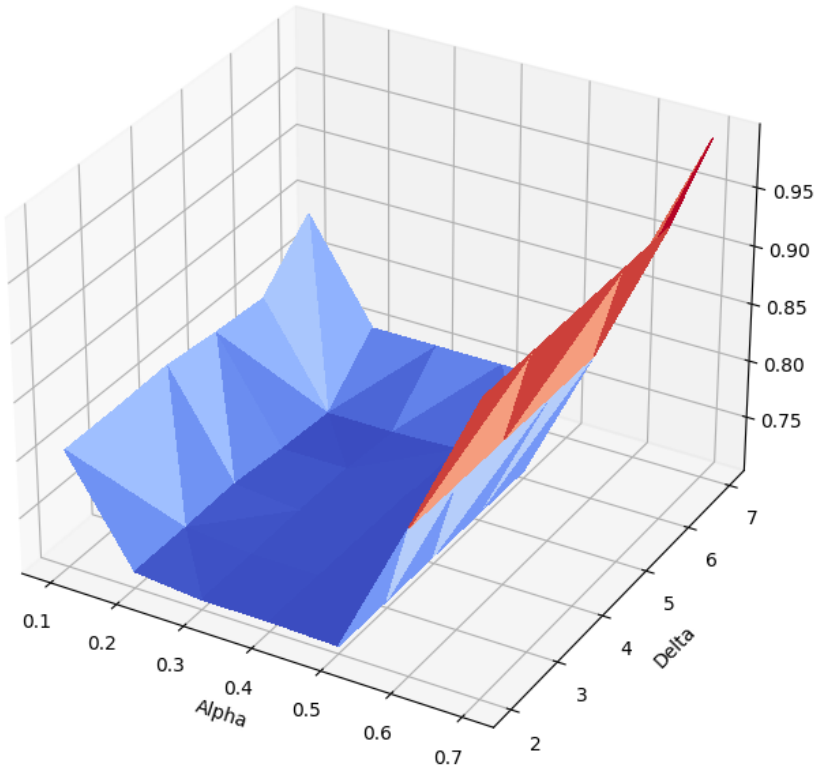

Figure A.1: The plot of the variation of the score (vertical axis) of the *FM spaCy V1* method from α and δ parameters.
The plot shows that the score remains almost stable for δ values between 2 and 6, but increases when delta is equal to 7. The variation of the score based on the α parameter is more significant and has a minimum score of 0.707839 with the α in the range of 0.2-0.5.

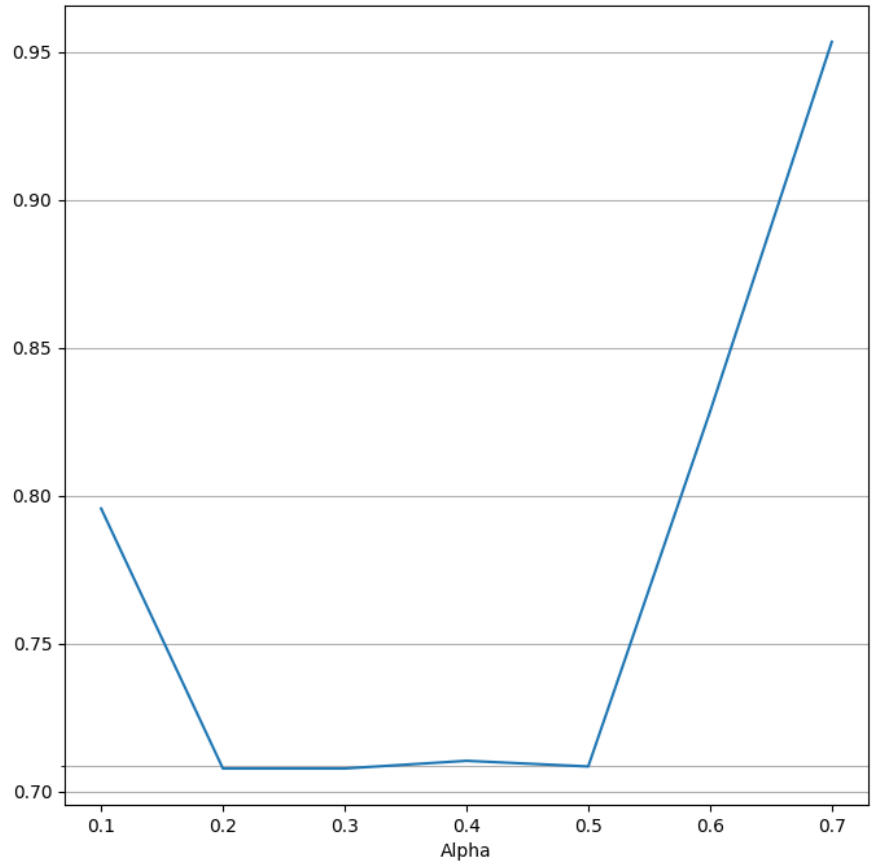

Figure A.2: The slice of plot reported in Figure A.1 with δ set to 6

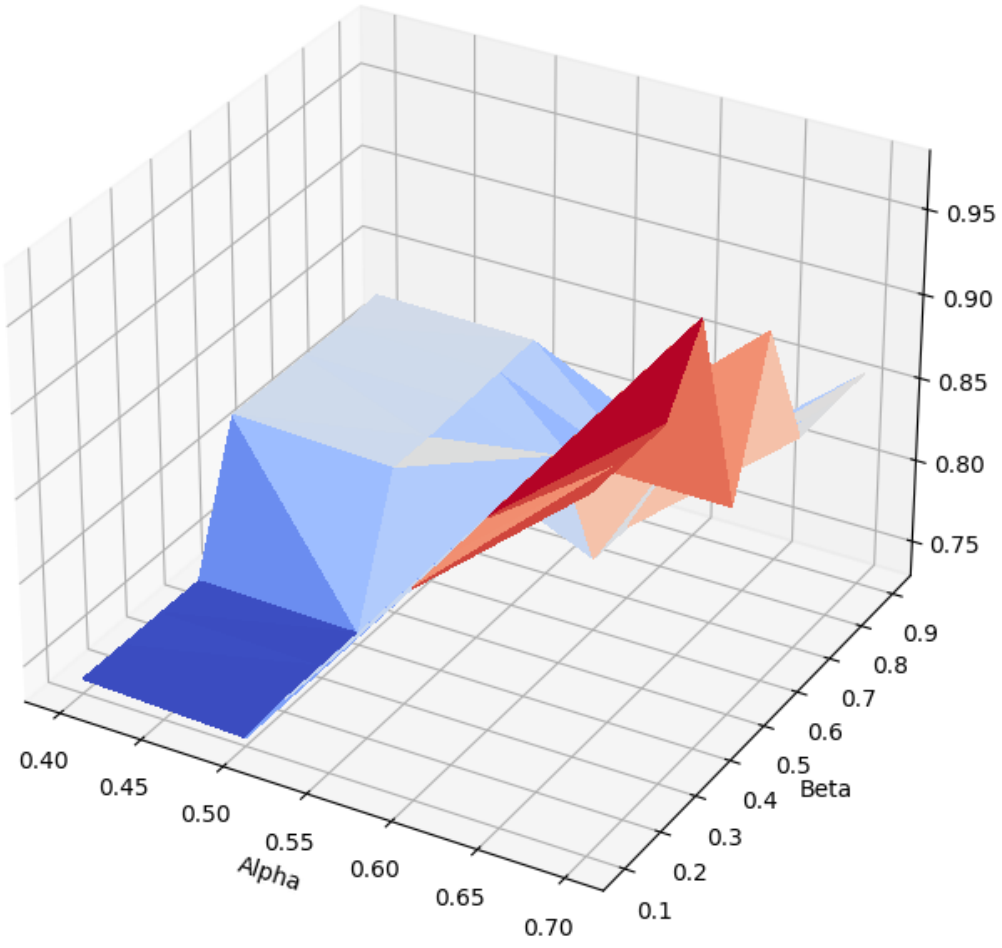

Figure A.3: The plot of the variation of the score (vertical axis) of the *FM spaCy V2* method from α and β parameters.
The plot illustrates that the score reaches a minimum of 0.733736 when β is lower than 0.5 and α falls within the range of 0.4-0.5. It is also evident that the optimal α value may vary depending on the chosen β parameter.

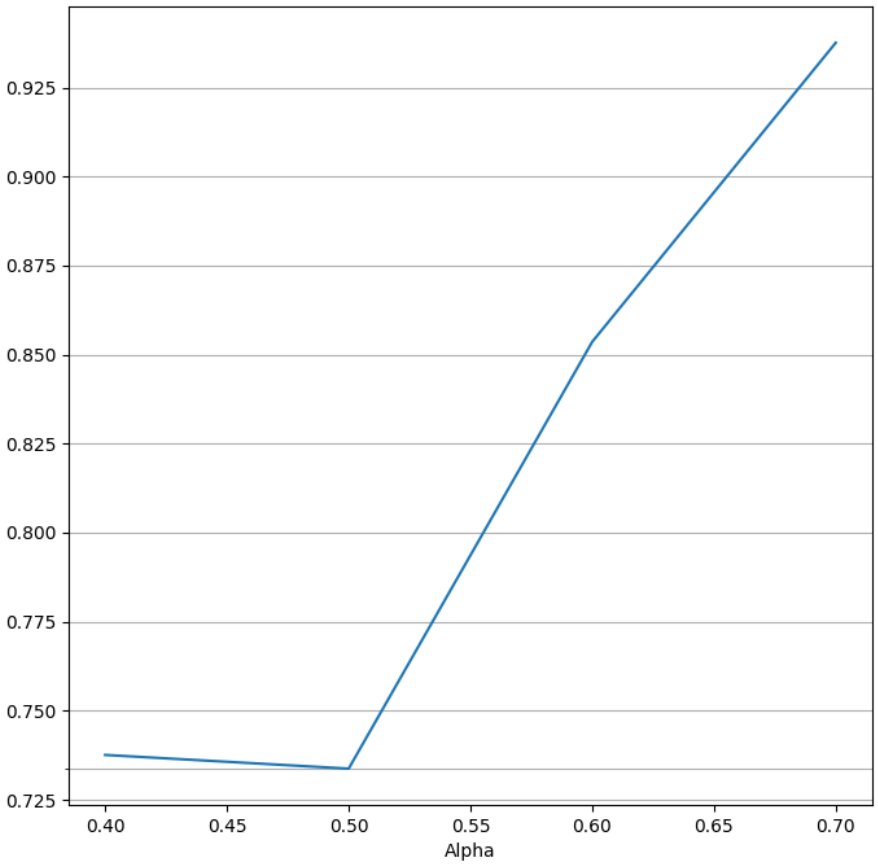

Figure A.4: The slice of plot reported in Figure A.3 with β set to 0.1. This slice has a minimum score of 0.733736 with an alpha of 0.5.

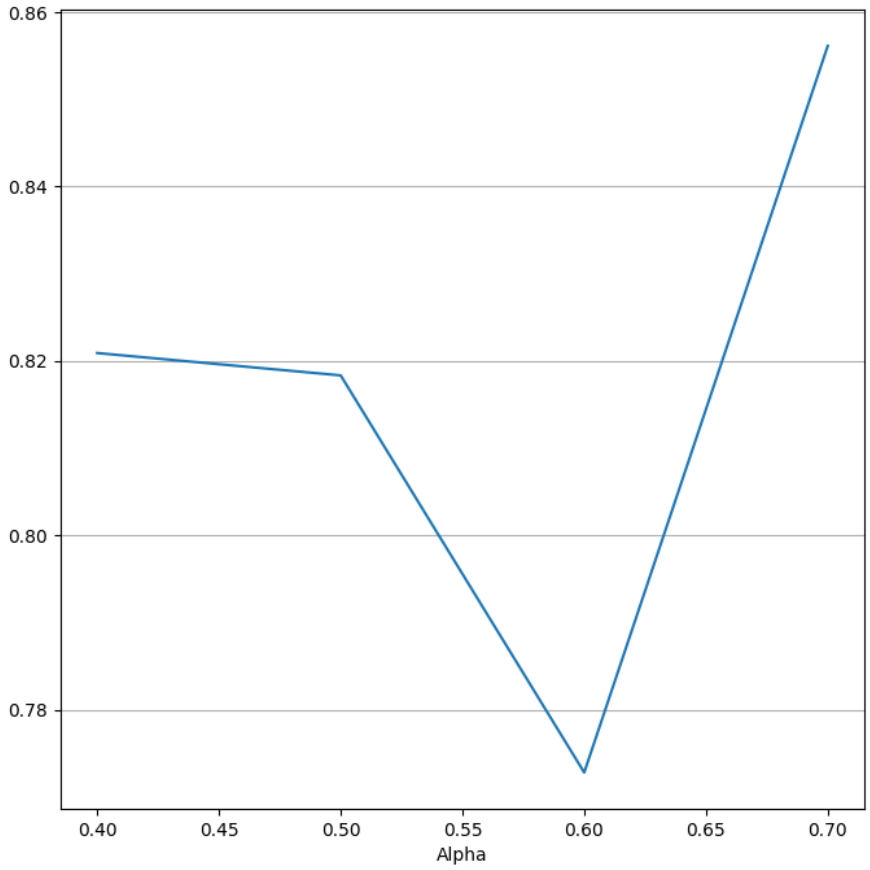

Figure A.5: The slice of plot reported in Figure A.3 with β set to 0.7 Unlike the slice in the Figure A.4, the minimum value of this slice has a score of 0.772809 with an alpha value of 0.6.

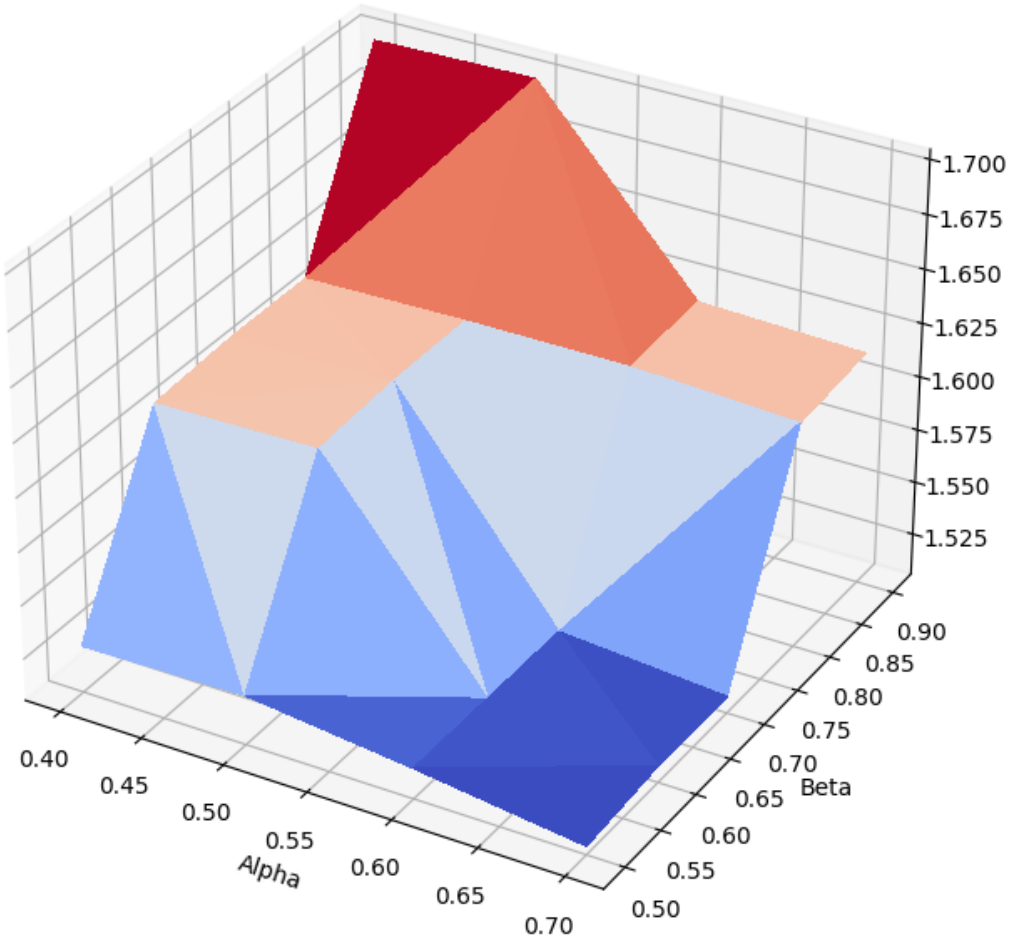

Figure A.6: The plot of the variation of the score (vertical axis) of the *FM spaCy V3* method from α and β parameters.
The plot indicates that the score reaches a minimum at beta 0.5 and alpha 0.7, with a score of 1.509540. The range of beta from 0.1 to 0.4 was not tested due to the assumption that the score would not decrease to an acceptable value, as it has an extremely high score compared to other methods and a minimum score in reported ranges higher than 1.

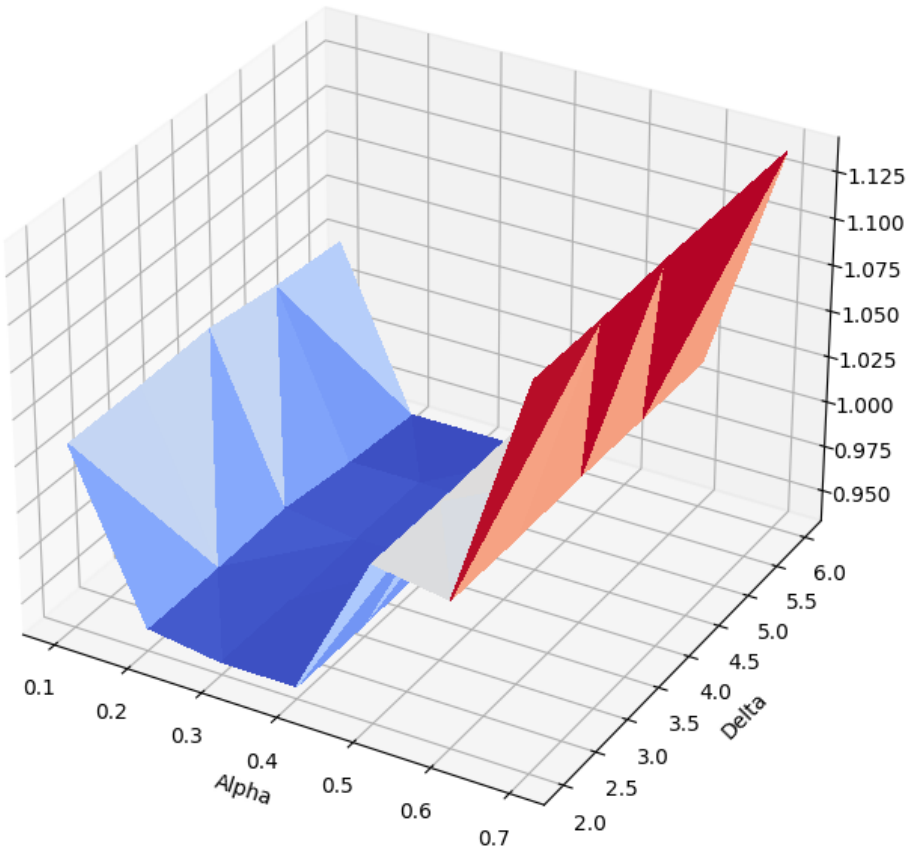

Figure A.7: The plot of the variation of the score (vertical axis) of the *FM regex V1* method from α and δ parameters.
The plot indicates that the score remains nearly constant for delta values ranging from 2 to 6. The score varies significantly based on the alpha parameter, with a minimum of 0.937059 in the alpha range of 0.2-0.4.

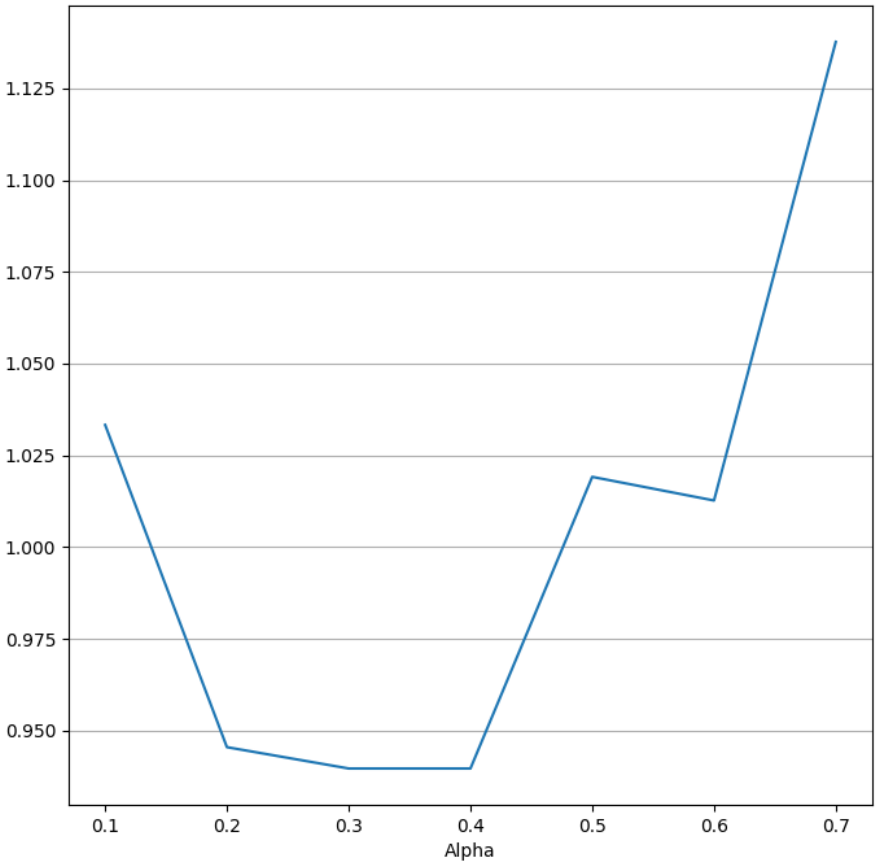

Figure A.8: The slice of plot reported in Figure A.7 with δ set to 3

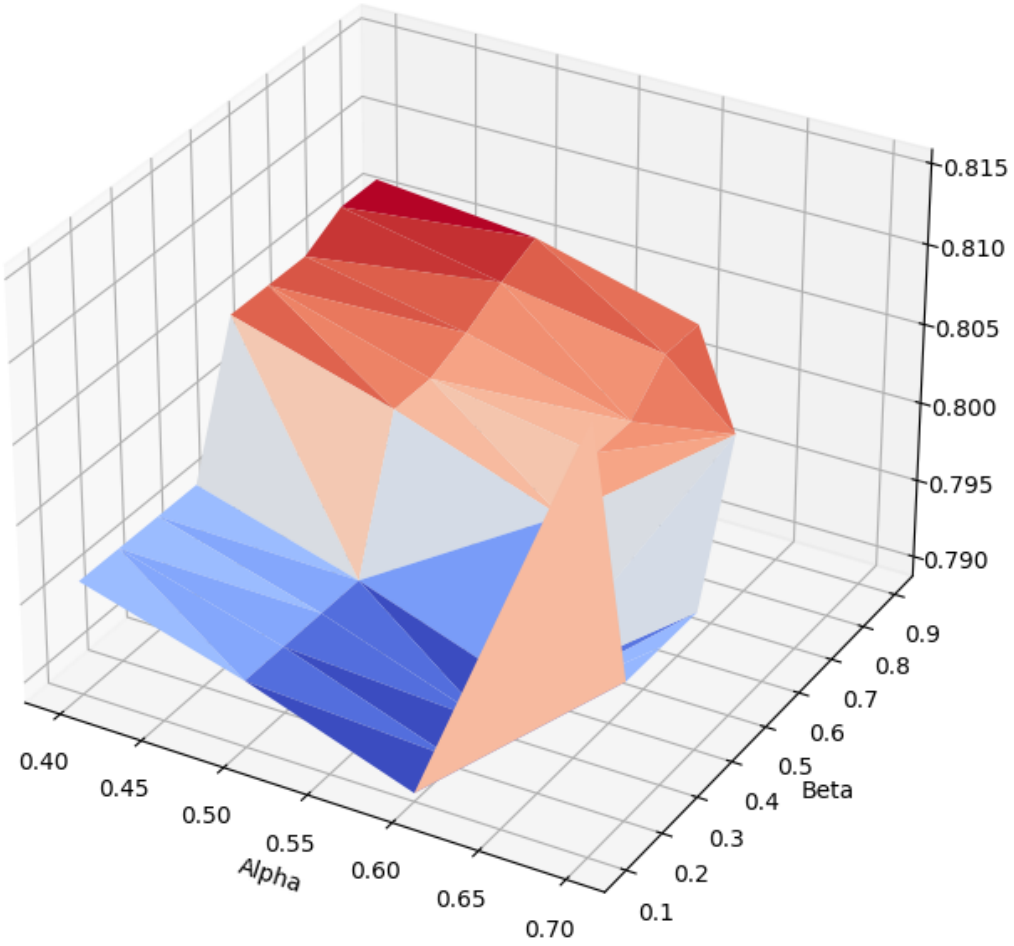

Figure A.9: The plot of the variation of the score (vertical axis) of the *FM regex V2* method from α and β parameters.
The plot illustrates that the score reaches a minimum of 0.789259 when beta is lower than 0.5 and alpha is 0.6. A noticeable slope is observed in the range of beta parameter 0.5-0.4. It is also important to note the unexpectedly high score of 0.815309 when alpha is equal to 0.7 and beta is 0.1.

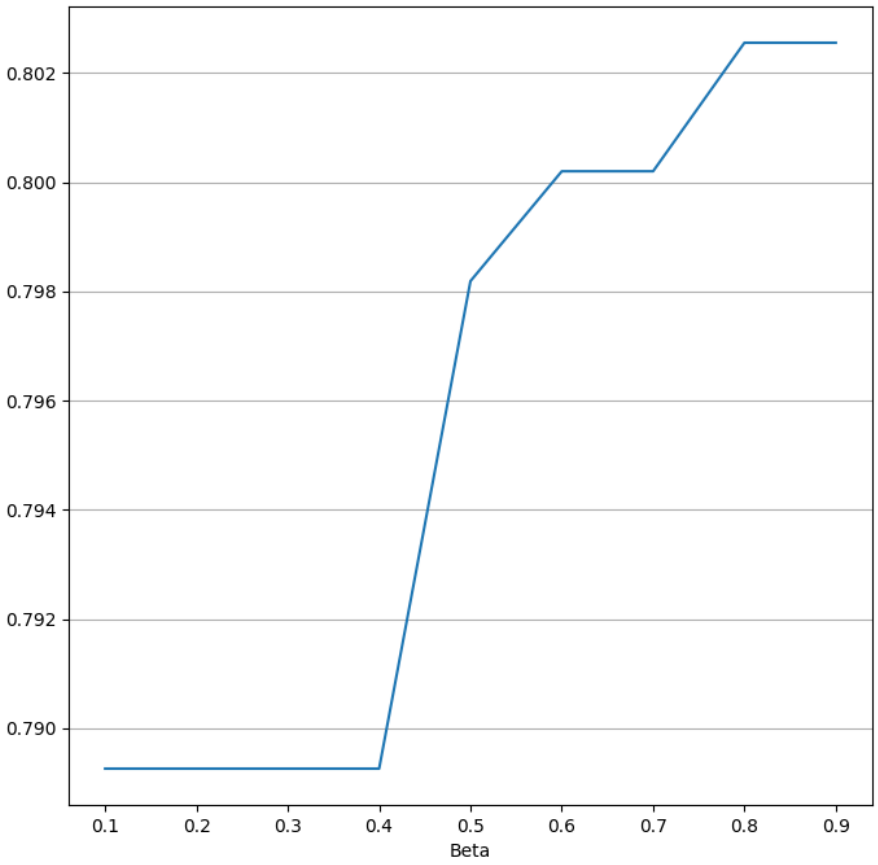

Figure A.10: The slice of plot reported in Figure A.9 with α set to 0.6

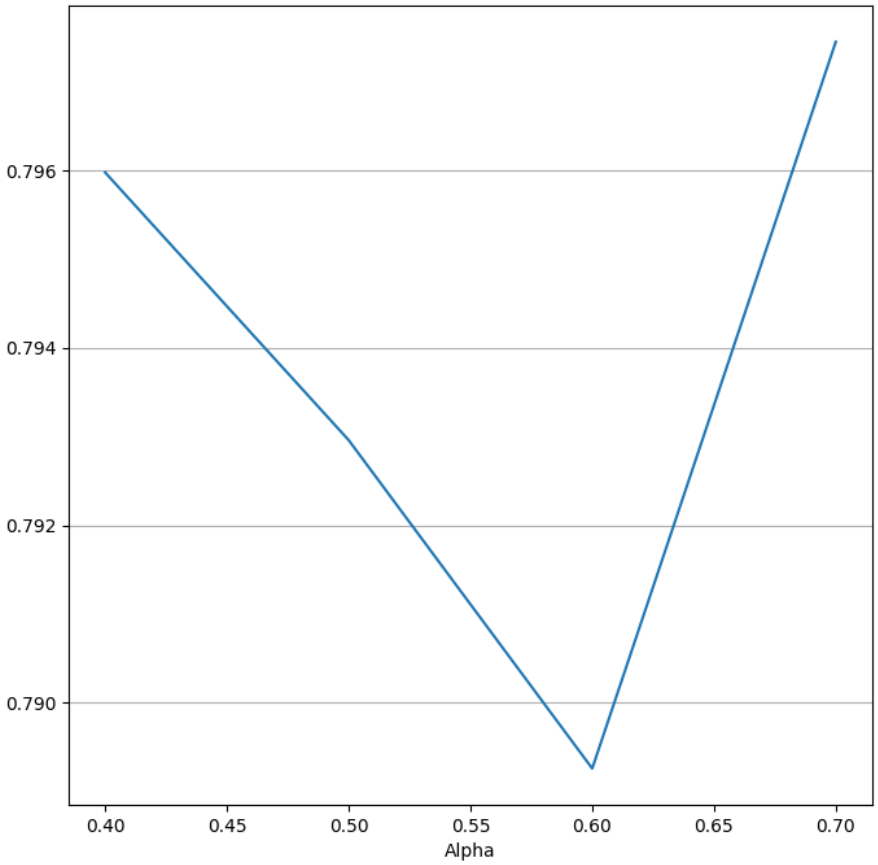

Figure A.11: The slice of plot reported in Figure A.9 with β set to 0.4. The plot appears similar when the beta parameter is set to 0.1, but the score is higher when the alpha value is 0.7.

Bibliography

[1] C. Leacock, M. Chodorow, M. Gamon, and J. Tetreault, *Automated grammatical error detection for language learners (Second edition., Vol. 25)*. Morgan & Claypool Publishers, 2014.

[2] Ćetković S., "The language of police reports: A quest for precision or a bureaucratic exercise of language degradation.," *Círculo de Lingüística Aplicada a la Comunicación, 71, 159-176*, 2017.

[3] M. D. Kernighan, K. W. Church, and W. A. Gale, "A spelling correction program based on a noisy channel model," in *COLING 1990 Volume 2: Papers presented to the 13th International Conference on Computational Linguistics*, 1990.

[4] A. Mykowiecka and M. Marciniak, "Domain–driven automatic spelling correction for mammography reports," in *Intelligent Information Processing and Web Mining* (M. A. Kłopotek, S. T. Wierzchoń, and K. Trojanowski, eds.), (Berlin, Heidelberg), pp. 521–530, Springer Berlin Heidelberg, 2006.

[5] K. H. Lai, M. Topaz, F. R. Goss, and L. Zhou, "Automated misspelling detection and correction in clinical free-text records," *Journal of biomedical informatics*, vol. 55, pp. 188–195, 2015.

[6] M. Chodorow and C. Leacock, "An unsupervised method for detecting grammatical errors," in *1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 2000.

[7] M. Gamon, "High-order sequence modeling for language learner error detection," in *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications* (J. Tetreault, J. Burstein, and C. Leacock, eds.), (Portland, Oregon), pp. 180–189, Association for Computational Linguistics, June 2011.

[8] M. Rei and H. Yannakoudakis, "Compositional sequence labeling models for error detection in learner writing," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (K. Erk and N. A. Smith, eds.), (Berlin, Germany), pp. 1181–1191, Association for Computational Linguistics, Aug. 2016.

[9] H. Yannakoudakis, T. Briscoe, and B. Medlock, "A new dataset and method for automatically grading ESOL texts," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (D. Lin, Y. Matsumoto, and R. Mihalcea, eds.), (Portland, Oregon, USA), pp. 180–189, Association for Computational Linguistics, June 2011.

[10] D. Colla, M. Delsanto, and E. Di Nuovo, "EliCoDe at MultiGED2023: fine-tuning XLM-RoBERTa for multilingual grammatical error detection," in *Proceedings of the 12th Workshop on NLP for Computer Assisted Language Learning* (D. Alfter, E. Volodina, T. François, A. Jönsson, and E. Rennes, eds.), (Tórshavn, Faroe Islands), pp. 24–34, LiU Electronic Press, May 2023.

[11] E. Volodina, C. Bryant, A. Caines, O. De Clercq, J.-C. Frey, E. Ershova, A. Rosen, and O. Vinogradova, "MultiGED-2023 shared task at NLP4CALL: Multilingual grammatical error detection," in *Proceedings of the 12th Workshop on NLP for Computer Assisted Language Learning* (D. Alfter, E. Volodina, T. François, A. Jönsson, and E. Rennes, eds.), (Tórshavn, Faroe Islands), pp. 1–16, LiU Electronic Press, May 2023.

[12] L. Bungum, B. Gambäck, and A. Brandrud Næss, "NTNU-TRH system at the MultiGED-2023 shared on multilingual grammatical error detection," in *Proceedings of the 12th Workshop on NLP for Computer Assisted Language Learning* (D. Alfter, E. Volodina, T. François, A. Jönsson, and E. Rennes, eds.), (Tórshavn, Faroe Islands), pp. 17–23, LiU Electronic Press, May 2023.

[13] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA, USA), pp. 282–289, Morgan Kaufmann Publishers Inc., 2001.

[14] M. Gamon, J. Gao, C. Brockett, A. Klementiev, W. B. Dolan, D. Belenko, and L. Vanderwende, "Using contextual speller techniques and language modeling for ESL error correction," in *Proceedings of the*

Third International Joint Conference on Natural Language Processing: Volume-I, 2008.

[15] D. D. Putra and L. Szabó, "UdS at CoNLL 2013 shared task," in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task* (H. T. Ng, J. Tetreault, S. M. Wu, Y. Wu, and C. Hadiwinoto, eds.), (Sofia, Bulgaria), pp. 88–95, Association for Computational Linguistics, Aug. 2013.

[16] Z. Yuan and T. Briscoe, "Grammatical error correction using neural machine translation," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (K. Knight, A. Nenkova, and O. Rambow, eds.), (San Diego, California), pp. 380–386, Association for Computational Linguistics, June 2016.

[17] Z. Xie, A. Avati, N. Arivazhagan, D. Jurafsky, and A. Y. Ng, "Neural language correction with character-based attention," 2016.

[18] S. Chollampatt and H. T. Ng, "A multilayer convolutional encoder-decoder neural network for grammatical error correction," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018.

[19] Y. Kementchedjhieva and A. Søgaard, "Grammatical error correction through round-trip machine translation," in *Findings of the Association for Computational Linguistics: EACL 2023* (A. Vlachos and I. Augenstein, eds.), (Dubrovnik, Croatia), pp. 2208–2215, Association for Computational Linguistics, May 2023.

[20] M. Loem, M. Kaneko, S. Takase, and N. Okazaki, "Exploring effectiveness of GPT-3 in grammatical error correction: A study on performance and controllability in prompt-based methods," in *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)* (E. Kochmar, J. Burstein, A. Horbach, R. Laarmann-Quante, N. Madnani, A. Tack, V. Yaneva, Z. Yuan, and T. Zesch, eds.), (Toronto, Canada), pp. 205–219, Association for Computational Linguistics, July 2023.

[21] F. Stahlberg and S. Kumar, "Seq2Edits: Sequence transduction using span-level edit operations," in *Proceedings of the 2020 Conference on*

Empirical Methods in Natural Language Processing (EMNLP) (B. Webber, T. Cohn, Y. He, and Y. Liu, eds.), (Online), pp. 5147–5159, Association for Computational Linguistics, Nov. 2020.

[22] V. Lyding, E. Stemle, C. Borghetti, M. Brunello, S. Castagnoli, F. Dell'Orletta, H. Dittmann, A. Lenci, and V. Pirrelli, "The PAISÀ Corpus of Italian Web Texts," in *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, pp. 36–43, Association for Computational Linguistics, 2014.

[23] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength Natural Language Processing in Python," 2020.

[24] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," *CoRR*, vol. abs/1911.02116, 2019.

[25] N. Team, M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard, A. Sun, S. Wang, G. Wenzek, A. Youngblood, B. Akula, L. Barrault, G. M. Gonzalez, P. Hansanti, J. Hoffman, S. Jarrett, K. R. Sadagopan, D. Rowe, S. Spruit, C. Tran, P. Andrews, N. F. Ayan, S. Bhosale, S. Edunov, A. Fan, C. Gao, V. Goswami, F. Guzmán, P. Koehn, A. Mourachko, C. Ropers, S. Saleem, H. Schwenk, and J. Wang, "No language left behind: Scaling human-centered machine translation," 2022.