*Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, pages 1849–1857
Marseille, 11–16 May 2020


# An In-Depth Comparison of 14 Spelling Correction Tools on a Common Benchmark

**Markus Näther**
University of Freiburg
79110 Freiburg, Germany
naetherm@cs.uni-freiburg.de

**Abstract**

Determining and correcting spelling and grammar errors in text is an important but surprisingly difficult task. There are several reasons why this remains challenging. Errors may consist of simple typing errors like deleted, substituted, or wrongly inserted letters, but may also consist of word confusions where a word was replaced by another one. In addition, words may be erroneously split into two parts or get concatenated. Some words can contain hyphens, because they were split at the end of a line or are compound words with a mandatory hyphen. In this paper, we provide an extensive evaluation of 14 spelling correction tools on a common benchmark. In particular, the evaluation provides a detailed comparison with respect to 12 error categories. The benchmark consists of sentences from the English Wikipedia, which were distorted using a realistic error model. Measuring the quality of an algorithm with respect to these error categories requires an alignment of the original text, the distorted text and the corrected text provided by the tool. We make our benchmark generation and evaluation tools publicly available.

**Keywords:** Benchmark, Evaluation, Spelling Error Correction, SEC, GEC

## 1. Introduction

Text continues to be one of the most popular formats for publishing information and knowledge, either in electronic or non-electronic form. Huge amounts of text are created every day and the variety of writing styles, formatting, structures, and publishing types are almost unlimited. Due to the fact that language is normally written by humans, it might be the case that different kinds of spelling and grammar errors occur within those written texts. For example, consider this sentence: *As an artist, Francis Bacon was a late starter* and this slightly distorted version: *As an artist, Framcis Bcon was a late starter*. The distorted version includes typical errors like hitting the key *m* instead of *n* and omitting a letter. While the human brain is quite good at fixing such errors during reading, the task is surprisingly difficult for computers. If one knows Francis Bacon, one would clearly identify the tokens *Framcis Bcon* as him. If one does not know him, one would probably detect that the token *Framcis* is not correct, since *Francis* is more familiar. The latter token *Bcon* could also stand for a wearable[1] and would hence be considered correct by many spelling correction tools although it does not make much sense in the given sentence.
Spelling correction is a fundamental problem in the field of natural language processing. Most algorithms and tools in the field simply assume that the given text is free from errors such as the above and they fail if this is not the case. In particular, this is true for basic problems such as POS-tagging, sentence parsing and entity recognition, as well as for more complex problems such as question answering (QA). For example, the QA system *Aqqu* (Bast and Haussmann, 2015) fails to produce an answer for the simple question *Who is Framcis Bacon* unless the typo in the question is corrected. The entity extraction tool Dandelion[2] suggests the entity *bacon* instead of the person *Francis Bacon*.

### 1.1. Overview

In Section 3. we give an overview of the benchmark, the occuring error categories, and how those errors are generated. In Section 4. we describe how the alignment between a given prediction and the groundtruth information is built and which metrics are evaluated. In Section 5. the results for all tools are presented and discussed.

### 1.2. Contributions

This paper is about the evaluation of existing spelling error correction tools and about the non-trivial tasks of constructing a benchmark, developing meaningful measurements and solving the underlying alignment problems. We consider the following as our main contributions.

- The construction of a benchmark of arbitrary size with detailed error category information, given arbitrary correct text.
- A set of criteria that allows us to measure not only the correction capabilities of a given tool but also whether a given error was detected but not corrected adequately.
- An extensive evaluation of 14 spelling error correction tools on our benchmark, according to our criteria. For each tool we provide a concise description of its strengths and weaknesses.
- We make the source code of our evaluation tools publicly available at `https://github.com/naetherm/SpellingBenchmark` and the visualisation results of the alignments accessible at `http://nsec.informatik.uni-freiburg.de/benchmark`.
- We discuss the steps which are necessary to build a fully satisfactory tool for spelling error detection and correction.

[1] `https://bcon.zone`
[2] `https://dandelion.eu/semantic-text/entity-extraction-demo`

## 2. Related Work

There has been extensive research on the problem of error correction, but we know of no widely used common benchmark. Each individual publication uses its own benchmark and compares to a (usually small) subset of methods (Tetreault et al., 2017; Lichtarge et al., 2019; Ghosh and Kristensson, 2017; Li et al., 2018). While (Ghosh and Kristensson, 2017) used the IMDB dataset and induced errors from the Twitter Typo Dataset[3], (Li et al., 2018) created a modified version of JFLEG (Tetreault et al., 2017), a dataset for grammar error correction. Brill et al. used a 10,000-word corpus (Brill and Moore, 2000) and other datasets like the ones of Roger Mitton[4] and Peter Norvig[5] only containing isolated errors, missing the context they occur in. This makes it hard to compare new approaches with already existing ones and to determine the kind of spelling errors an application is capable of correcting and the ones which it has problems with. The main motivation behind this paper and the benchmark was to evaluate the different approaches in a common framework. We developed a general method for the generation of benchmarks that has knowledge about specific error categories and how to create them. Through this we can generate the input source and the corresponding groundtruth files as JSON, with the input source and tools for transformation being available for download.

There exists a large number of spelling error correction tools. A simple search provides countless hits for either standalone applications, browser plugins and addons, or online services, either as free or paid service. The variety in available tools is confusing and there does not seem to be a clear winner, especially because the normal tool description lacks information about which kind of errors can be detected and fixed at all. So far, there has been no benchmark for this problem and no comprehensive evaluation of existing systems is available at all, which is surprisingly, given the practical importance of the problem. Due to that this paper wants to bring some clarity into the jungle of available tools and provides a clear measurement for different types of errors.

## 3. Benchmark

This section is about the generation of a benchmark test set from a given text source. In the subsequent sections we will deal with (1) the error categories that can be distinguished in Section 3.1., (2) the used source dataset which will be induced with errors in Section 3.2., and (3) how spelling errors are induced in the datasets in Section 3.3..

### 3.1. Error Categories

To the best of our knowledge no publication about spelling error correction has knowledge about the underlying errors within the used datasets and benchmarks. We build our evaluation with the existence of different error categories in mind and made it possible to not only measure when an error was adequately corrected but also when an erroneous word was detected as such and was tried to be corrected, independent of the actual correction. As a starting point we came up with the following error categories.

**NON_WORD**. Can be divided in *pseudo words*, representing a sequence of letters following the graphotactic and phonotactic rules of a particular language with no meaning in that language, and *non words* which violate those rules like "*taht*". We do not distinguish between those two categories and simply mark them as NON_WORD.

**REAL_WORD**. Real-words represent words that were replaced by other words of the current language, like "*form*" instead of "*from*".

**HYPHENATION**. Words in visual text can be hyphenated, which is frequent for formats with multiple columns. This case deals with hyphens within a word, through wich tokens are "broken up" in two parts at different positions within a text, e.g "*hy-phenation*".

**COMPOUND_HYPHEN**. Some words are compositions of two words, making a hyphen mandatory. For those words it is uncommon to either remove that hyphen or split such a compound word in two subsequent words, e.g. "*high quality*" instead of "*high-quality*".

**CAPITALISATION**. Some words in visual text contain more context and meaning, like organisation names like *Apple™* which should not be confused with the fruit *apple*. For this case the capitalisation is a crucial indication of whether a word represents a noun or a proper noun.

**ARCHAIC**. Some languages like German and French had spelling reforms during the last decades, changing the way words are written. In some reforms the grammar was changed too, e.g. in the french spelling reform of 1990. Although the used benchmark has no archaic words we wanted to include it here for the sake of completeness.

**SPLIT**. A word can, due to several reasons, get split up in two distinct words. The word can thereby get split according to rules of syllables or randomly at any position.

**REPEAT**. Double words are uncommon in some languages like english, where only a hand full of different cases of doubled words exists, e.g. the form "had had" and a double-is. Nevertheless, they are more common in other languages making it an important error category.

**CONCATENATION**. Concatenation errors are a popular source of errors, not only by a user who does not hit the space key but also by e.g. OCR tools or pdf text extraction tools. Thereby two words are combined to a single word, either resulting in a non existing word or a changed context.

**PUNCTUATION**. Punctuation describes the arrangement of marks among words in a sentence, enabling comprehension. Problems with punctuation are starting with wrong sentence endings (e.g. a point instead of question mark for questions) and go other to misplaced commas, which can change the meaning of a sentence.

**MENTION_MISMATCH**. Whenever a personal pronoun is used there exists the possibility that, due to various reasons, the correct personal pronoun is replaced by another one, e.g. instead of *he* on types *she*.

**TENSE**. Verbs occur with different tenses, including regu-

[3] `http://luululu.com/tweet/`

[4] `https://www.dcs.bbk.ac.uk/~ROGER/aspell.dat`

[5] `http://norvig.com/ngrams/spell-errors.txt`

lar and irregular tenses for a small subset of verbs. Especially for the conjugation of verbs it is hard to distinguish between grammar and spelling errors. On one hand one may apply the wrong conjugation rules on an irregular verb, e.g. *split, splitted*.

### 3.2. Dataset

To receive a huge and diverse set of written language we decided to take the English Wikipedia[6] as text source. Wikipedia is an online encyclopedia containing many articles about diverse topics. From the received compressed archive of Wikipedia we extract the single articles and remove unnecessary content like images, links, templates, and markdown notations. In the end we have a cleaned up corpus which just contains sentences. For the extraction of the downloaded Wikipedia Dumps we are using the already available tool WikiExtractor[7] and extended it with functionality serving our needs.

### 3.3. Error Generation

The generation of noised sequences is driven by a set of different generators each containing an equally distributed calling probability. Basic generators are responsible for introducing random insertions, deletions, replacements, or swapping of single characters within a randomly chosen word. With those strategies being very uncommon we further added a Gaussian keyboard, simulating typos occuring during the process of typing, e.g. by hitting a key in the local neighbourhood. Hyphenation errors occur in three scenarios, either through syllabification integrated by the hyphenation algorithm of Liang (Liang, 1983), creating compound words like *high-tech*, or creating two separate words instead of the compound one, e.g. *high quality* instead of *high-quality*. The wrong usage of quotation marks for citations, missing commas, or wrong placement of those all belong to the categories of punctuation errors. Wrongly placed commas can typically occur before conjunction words, like *and, while, if*, for which an own generator was introduced. Mention mismatches were introduced by replacing occurences of pronouns like *she* with their counterparts, e.g. *he*, which, although also representing a REAL_WORD error too, are marked as MENTION_MISMATCH. Tense errors are generated by identifying verbs and - if available - replacing them with another tense of that particular verb. Capitalisation errors are introduced by swapping the case of the first character of the currently observed word, in conjunction with the casing of the remaining word. For all of the above mentioned strategies, especially the basic generators, we further have to distinguish whether the created word is either of type NON_WORD, REAL_WORD, CAPITALISATION, and HYPHENATION done by analysing the created noised token. For the introduction of REAL_WORD errors we additionally introduced a language dictionary, populated with all words of the Oxford Dictionary[8], which has knowledge about the neighbouring words according to the Levenshtein edit distance and phonetic similarity. This neighbourhood is defined as $neighbours(w_i, w_j)$ in Equation 1, with $w_i$ and $w_j$ representing words of a language domain $\mathbb{L}$, $ED()$ is representing the Levenshtein distance, and $SIM()$ represents the usage of the double metaphone phonetic encoding algorithm (Philips, 2000). $\tau$ and $\sigma$ are constant thresholds set during the generation of the language dictionaries, with $\tau = 2$ and $\sigma = 3$.

$$neighbours(w_i, w_j) = \begin{cases} T & ED(w_i, w_j) <= \tau \vee \\ & SIM(w_i, w_j) <= \sigma \\ F & \text{else} \end{cases} \quad (1)$$

[6] https://en.wikipedia.org

[7] https://github.com/attardi/wikiextractor

[8] https://oed.com

## 4. Evaluation

### 4.1. Prediction Format

Although it would be possible providing corrections as raw text we favoured a structured format, allowing us to add additional information each of the evaluated tool can provide like the detected error type or a sorted list of correction suggestion candidates. For each token the format provides a unique id (containing an article, sentence, and word identifier representing the position), the most probable corrected token, and optional information about the detected error and further correction candidates.

### 4.2. Token Alignment

Aligning source, groundtruth, and prediction sequences is split up in two seperate stages, the compiling and linking which will be discussed in detail in the subsequent sections. For better understanding of how tokens are aligned one may follow the visualised alignment in Figure 1. A token is usually more than raw text as presented in Figure 1, a token is a tuple `τ = (text, idx, err, PPos, SPos, GPos)`, with $text$ being the text, $idx$ the position index within the groundtruth sequence, $err$ describes the error category, and $PPos, SPos, GPos$ being the indices of the prediction, source, and groundtruth tokens associated with this token. The groundtruth token representing the $a$ from the example in Figure 1 would have the following form: (*"a"*, 4, REPEAT, [], [4, 5], [4]), with the indices information for $PPos$ being empty at that point.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **S** | The | 20-yearold | Julia | becmme | a | a | lawyer | in1976 | . |
| **P** | The | 20 | year | old | Julia | become | a | lawyer | in | 1976 | . |
| **G** | The | 20-year-old | Julia | became | a | lawyer | in | 1976 | . |

Figure 1: The tokenised input sequences for source (S), prediction (P), and groundtruth (G); for simplicity we only visualise the text of the tokens.

#### 4.2.1. Alignment Compiling

Sequence compilation is done separately for each pair of (prediction, source) and (prediction, groundtruth). The purpose is the identification of identical blocks occuring in both sequences, using a distance-sensitive $n$-gram approach to find the longest common subsequences (LCS), and the remaining blocks of unmatched tokens, which are also the results of this operation. For each sentence pair of the input sequences we add the tuple of tokenised, lower-cased sentences $(S, T)$ to a queue **Q** and repeat the following until **Q** is empty:

1. Given two sequences $S = (s_1, s_2, \ldots, s_g)$ and $T = (t_1, t_2, \ldots, t_h)$ by $\mathbf{Q}$, with $s_i$, $t_j$ representing the tokens of the sequences, define the maximum distance between both sequences as $d = |g - h|$ and the length of the starting $n$-gram with $n = min(g, h)$.

2. Build the set of all $n$-grams $\hat{S} = \{[s_{1:n}], [s_{2:n+1}], \ldots, [s_{g-n+1:g}]\}$ and $\hat{T}$ respectively, with $[x_{a:b}]$ being the list of tokens $x_a$ to $x_b$.

3. $\forall \hat{s}_k \in \hat{S}$ check if $\exists \hat{t}_l \in \left[\hat{t}_{k-d}, \ldots, \hat{t}_{k+d}\right] \subseteq \hat{T}$ so that $\hat{s}_k = \hat{t}_l$, where $\hat{s}_k = \hat{t}_l := \forall e \in \{1, \ldots, n\} |\ \hat{s}_{k_e} = \hat{t}_{l_e}$ is a comparison of the raw text of these tokens.

4. (a) If two identical $n$-grams $\hat{s}_k$ and $\hat{t}_l$ were found the tokens of these $n$-grams defining the block are added to the list of identical blocks and the tuples of the remaining unmatched tokens $([s_{1:k-1}, t_{1:l-1}])$ and $([s_{k+n:g}, t_{l+n:h}])$ are added to $\mathbf{Q}$, if the tuple is not empty.

   (b) If no identical $n$-grams were found set $n = n - 1$ and start over at (2), until $n = 1$ and no alignment was found. In this case all tokens of $S$, and $T$ are added to the set of unmatched blocks.

After this operation was performed on both sets we have the alignment information and the remaining unmatched tokens as visualised in Figure 2. With this alignment information we can build cross-links due to the connectivity-knowledge between groundtruth and source. Iterating over both sets of tokens, the groundtruth and source, we validate whether the corresponding tokens contains assignment information not present for the currently observed token.

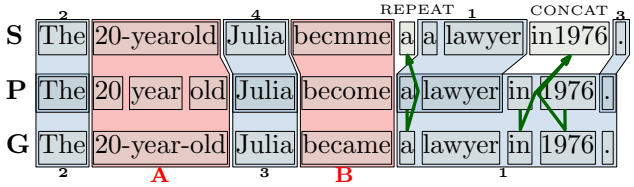


Figure 2: Tokens shaded in light blue are aligned blocks. The blocks are labeled in the order in which they were found is. We will refer to the unmatched groups as **A** and **B** during the linking step. Tokens which are connected by a green arrow are resolved by the connectivity-knowledge between groundtruth and source. The remaining areas, shaded in red, represent the still unaligned tokens between the sequences.

#### 4.2.2. Alignment Linking

Before continuing we group the lists of unmatched tokens between (prediction, groundtruth) and (prediction, source) to (prediction, groundtruth, source) triples, as it is visualised in Figure 2, but keep track about (prediction, groundtruth) and (prediction, source) as $G_{PG}$ and $G_{PS}$ respectively. For each group within $G_{PG}$ and $G_{PS}$ there are three different cases with $|x|$ and $|y| > 0$: (1) **x**-to-**y**, (2) **x**-to-zero, and (3) **x**-to-none.

**x-to-y.** This is one of the most common cases and is the first case for both **A** and **B** of Figure 2. For two given sequences $S = (s_1, s_2, \ldots, s_g)$ and $T = (t_1, t_2, \ldots, t_h)$ the following checks are performed and two tokens $s_i$ and $t_j$ are only aligned if one of the 9 conditions hold, with $\lambda = 1$, $mL = min(s_i, t_j)$. $sim$ in Equation 2 calculates a similarity score between $0-1$ of the tokens $t_i$ and $t_j$ with $jw$ being the Jaro-Winkler similarity and $ED$ the Levenshtein edit distance. $|t_i|$ will return the number of characters of that token.

$$sim(t_i, t_j) = \frac{\left(jw(t_i, t_j) + \left(1 - \frac{ED(t_i, t_j)}{max(|t_i|, |t_j|)}\right)\right)}{2} \quad (2)$$

We used a combination of both due to the stability of Jaro-Winkler regarding typos in small words, e.g. when calculating the similarity between *taht* and *that* $jw$ will return a value $> 0.9$ while $ED$ will be 2, resulting in a value of $0.5$ for the corresponding normalisation term. Tokens in $\lfloor\rfloor$ are the lower-cased variants, \$ the index of the last character of a token, and $s_{i;k:l}$ is the notation for the substring of $s_i$ from position $k$ to $l$, with $k < l$:

1. $sim(\lfloor s_i \rfloor, \lfloor t_j \rfloor) > 0.7$
2. $t_j = s_{i;0:\$-\lambda}$
3. $s_i = t_{j;0:\$-\lambda}$
4. $sim(\lfloor s_i \rfloor, \lfloor t_{j;0:\$-\lambda} \rfloor) > 0.7$
5. $sim(\lfloor s_{i;0:mL} \rfloor, \lfloor t_{j;0:mL} \rfloor) > 0.7$
6. $t_j = s_{i;\$-\lambda:\$}$
7. $s_i = t_{j;\$-\lambda:\$}$
8. $sim(\lfloor s_{i;\$-\lambda:\$} \rfloor, \lfloor t_{j;\$-\lambda:\$} \rfloor) > 0.7$
9. $sim(\lfloor s_{i;\$-mL:\$} \rfloor, \lfloor t_{j;\$-mL:\$} \rfloor) > 0.7$

**x-to-zero.** $x$ tokens of $S$ cannot be aligned to any tokens of $T$. Though we know where the gap in $T$ exists to whose index position $i$ we refer to as $\check{t}_i$. For the left-most token of $x$ and $\check{t}_{i-1}$ evaluate (1) and (6-9), respectively evaluate (1-5) for the right-most token of $x$ and $\check{t}_{i+1}$. If no correspondence was found the investigated tokens of $x$ are unmatchable at the current state.

**x-to-none.** The tokens **x** are not aligned to either a token

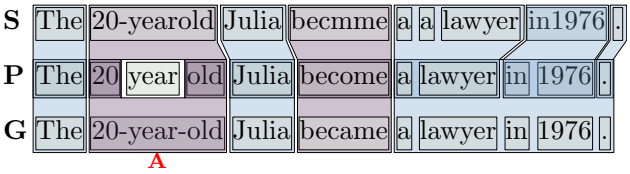


Figure 3: The token of **B** and the two outer ones of **A** in **P** are aligned, except the token *year*. This one has no tokens or empty gaps in **S** nor in **G**. The last possibility is that it occurs within already aligned tokens, check by **x-to-none**.

nor an empty gap within the other sequence. An example of that can be seen in Figure 3 were the token *year* in **P** has no connection to any other token, nor is assignable to an empty gap. We refer to the tokens surrounding the tokens $x$ as

$x_{-1}$ and $x_{\$+1}$. For these tokens we collect the groundtruth and source indices in $\check{G}_{pos} = \{x_{-1;GPos}, x_{\$+1;GPos}\}$ and $\check{S}_{pos} = \{x_{-1;SPos}, x_{\$+1;SPos}\}$. Then check $\forall x_i \in x$ if $\exists \check{g}_j \in \check{G}_{pos}$ so that $x_i \subset \check{g}_j$, with $x_i \subset \check{g}_j \coloneqq x_i = \check{g}_{j;k:l}$ for $k < l$.

1. If that holds for a pair $x_i$, $\check{g}_j$ those tokens are aligned, $x_i$ is removed from the set, and the cross-links are updated.

2. Otherwise $x_i$ is unmatchable to any token.

The same is done for $\check{S}_{pos}$ respectively.

### 4.3. Benchmark

Two separate test sets were created, with the first one containing no error and the second one all of the previous mentioned errors.

**Stability Test.** Checking how tools act on error free sequences we only evaluated the word and sequence accuracy for this test set. The set consists of 241 sentences with 6362 words in total.

**Test set.** Due to the limitations of some tools to just a few sentences per day or in total we limited the set to $\approx$ 800 sentences, while tools that can be evaluated automatically used a test set of $\approx$ 9060 sentences. Only 2,263/19,950 tokens are assigned to one of the previously mentioned error categories, so that approx. $11\%/8\%$ of the tokens contain one error. Setting the random seed to 42 it is guaranteed to always generate the same subset, containing the exact same errors, when using the same Wikipedia dump files.

### 4.4. Evaluation Metrics

Through the evaluation we are collecting a wide range of information which will be described in more detail below and how those values are determined.

**Word Accuracy**. For the tokens $T$ of a sequence $s_i$ the word accuracy is given by $WA : \forall t_i \in T : \frac{c(t_i)}{|T|}$, with

$$c(i) = \begin{cases} 1 & |t_{i;GPos}| = |t_{i;PPos}| \wedge \\ & \forall j \in |t_{i;GPos}| \Rightarrow t_{i;GPos \rightsquigarrow j} = t_{i;PPos \rightsquigarrow j} \\ 0 & \text{else} \end{cases}$$

where $t_{i;PPos}$ are the prediction indices of token $t_i$ and $t_{i;PPos \rightsquigarrow j}$ the access to the text of the prediction token at position $j$, and respectively for $GPos$.

**Sequence Accuracy**. The sequence accuracy for all sequences $S$ is given by $SA : \forall s_i \in S : \frac{s(s_i)}{|S|}$, with

$$s(s_i) = \begin{cases} 1 & WA(s_i) = 1 \\ 0 & \text{else} \end{cases}$$

**Error Category Detection**. Although most of the determined tools do not have an evaluation of the underlying error category the measurements for the detection can be determined indirectly by the correction itself. *Prediction* is measured as the proportion of adequately detected errors of each error category by a tool over all articles. *Recall* as the proportion of the errors of each error category marked in the groundtruth of all articles that were detected during the correction. *F-Score* is defined as the harmonic mean of precision and recall. With the majority of tools not supporting error categories we measure the detection through their error correction. If a token was changed with respect to the groundtruth it cannot be *NONE* anymore. If it was adequately corrected we assume that the error category was correctly detected. Otherwise the proposed corrected token is analysed to investigate which category is the most probable, e.g. investigate whether the character of the first letter was changed indicating capitalisation errors.

**Error Category Correction**. We further evaluate independently whether a given token was adequately corrected. *Precision* is thereby measured as the proportion of corrected errors for an error category by a tool in all articles. *Recall* is measured as the proportion of the errors of each error category marked in the groundtruth of all articles that were adequately corrected. *F-Score* is defined as the harmonic mean of precision and recall. Unaligned prediction tokens after linking will be counted as REAL_WORD errors at this point.

**Suggestion Adequacy (SA)**. SA measures the adequacy of the proposed suggestions, if provided by the tool. In accordance to (Starlander and Popescu-Belis, 2002) we are using the following metric $SA$ for measuring the adequacy, with $t$ being the current token and $s$ a list of the predicted token and further suggestions, if provided:

$$SA(t, s) = \begin{cases} 0 & s = \emptyset \\ 1 & t = s[0] \\ 0.5 & t \in s \\ -0.5 & \text{else} \end{cases}$$

The overall suggestion adequacy $\bar{SA}$ is determined by normalising the sum of all scores by the number of tokens. With $T$ being the set of predicted tokens defined as (predict, sugg=[], err=[]), where *predict* is the predicted tokens, *sugg* a list of alternative predictions, and *err* the proposed error category.

$$\bar{SA} = \frac{\sum_{k=1}^{T} SA(s_{k;predict})}{|T|}$$

**Overall Performance**. In general one is interested in a single-valued overall performance measure of a tool when compared to others. A first approach was made by (Starlander and Popescu-Belis, 2002) who measured the predictive accuracy (PA) of a spelling correction tool as the likelihood of a token, correct or incorrect, being handled accurately by the spelling checker. This metric does not make any prediction about the performance of a spelling error correction tool, as it rather reflects the performance of a spelling checker in a given evaluation. Although this measurement gives a good overall measurement for the competency of a spelling correction system, the problem with this type of measurement is that the difference between an adequate and an inadequate spelling correction system is relatively small. Due to that (Starlander and Popescu-Belis, 2002) refined the measurement, using the harmonic mean between recall and

precision, entailing that tools making use of trivial strategies, for example, scoring all tokens as correct to obtain high lexical recall, will be penalised in terms of the error recall. The mean score is a measure which calculates an average that is lower than the normal average score. The mean tends to be more severe on lower scores, awarding a score which is closer to the lower score than the normal average score would be. As pointed out, these measures are especially valuable to detect trivial strategies. These metrics does not take information about specific error categories into consideration, which leads to a general integration of evaluation metrics taking the general correction performance over all error categories into consideration, which can be seen in Equation 3 and 4. $E_{Score}$ is an averaged correction performance over all error categories, including the category NONE which represents that no error is present. $P_{Score}$ uses the correction of those NONE as a penality parameter. $N$ represents a list of error categories, starting with $NONE$ as the first entry for simplicity. $c_j$ is the number of adequately corrected errors of error category $j$ and $t_j$ is total amount of errors for the category $j$. For $P_{Score}$ the correction performance of NONE is multipled with the averaged performance over all error categories, due to the fact that we want a changed output for all error categories instead of the NONE category where we explicitely do not want any changed output. Therefore the correction parameter for NONE can be interpreted as a penality parameter.

$$E_{Score} = \frac{\sum_{i=0}^{|N|} \frac{c_i}{t_i}}{|N|} \quad (3) \qquad P_{Score} = \frac{\sum_{i=1}^{|N|} \frac{c_i}{t_i}}{|N| - 1} \frac{c_0}{t_0} \quad (4)$$

## 5. Results

Within this section we shortly introduce all tested tools and then show and dicuss the results on the previously in Section 4.3. introduced benchmarks.

### 5.1. Tools

We evaluated the following 14 spelling error correction tools. An overview and comparison of their features, according to available information, is given in the description. Tools that are marked with $^+$ are paid services which can include a short/free trial version with possibly limited features, while all other tools and library are available for free. Tools marked with $^*$ were only able to process the input token-wise.

**BingSpell**$^+$ is part of the Microsoft Cognitive Toolkit, being capable of correcting word breaks, slang words, homonyms, names, and brand names (Microsoft Corp., 2019).

**LanguageTool**$^+$ is capable of detecting grammar errors, general spelling errors, capitalisation errorss, words breaks, and hyphenation errors as well as slang words (LanguageTool.org, 2019).

**GrammarBot**$^+$ is a dictionary based approach for correcting found spelling errors (GrammarBot, 2019).

**JamSpell** is a spell checking library, that checks the surrounding of a given word to make better predictions and can process up to 5,000 words per second (Ozinov, Filipp, 2019).

**TextRazor**$^+$ is capable of detecting typos in real world entities like people, places, and brands, although they were never seen by the algorithms behind TextRazor. Additionally it is capable of identifying incorrect homophones and the surrounding context of words to advance the correction capabilities and correction accuracy (textrazor.com, 2019).

**PyEnchant** is a Python package, providing binding to the Enchant spellchecking library (Ryan Kelly, 2019).

**HunSpell**$^*$ is one of the default spelling error correction tools that is also used in tools like LibreOffice, OpenOffice, Firefox, etc. (László Németh, 2019).

**Aspell**$^*$ is the default spelling error correction tool for all GNU operation systems, designed to replace Ispell (Kevin Atkinson, 2017).

**Google** contains a spelling correction tool for all services. For evaluation we used *Google Docs* as pipeline for correcting errors within the given source files (Google, 2019).

**Grammarly**$^+$ is a web service capable of checking grammar and spelling. Thereby it can detect grammar, spelling, punctuation, word choice, and style mistakes following common linguistic prescription (Grammarly Inc., 2019).

**Long-Short-Term-Memory (LSTM)** is not a tool by itself but belongs to the toolset of machine and deep learning and it especially used for sequence to sequence tasks (Hochreiter and Schmidhuber, 1997). We used a network with 2 LSTM layers for an unidirectional encoder and decoder each, both with en embedding dimension of 256.

**Transformer** is a specific sequence to sequence neural network architecture, as described in (Vaswani et al., 2017). For our purposes we retrained Transformer on a character-level using our training set, with the network configuration proposed by (Vaswani et al., 2017).

Two baselines were included: (1) a dictionary based approach based on Peter Norvig[9] (*Norvig*$^*$) and (2)) a character-level $n$-gram (*N-Gram*$^*$) approach with $n = 3$. The training set for LSTM and Transformer, both implemented using *fairseq*[10] – generated through the previously described generator – does not include any of the articles used for the benchmark and used different error generators than those used for the test set generation.

### 5.2. Evaluation Results

**Stability Test**

The results on the stability testset are reported in Table 1. For completeness we reported the performance of $WA$, $SA$, and $E_{Score}$. Tools with the best word accuracy (WA) are Google, BingSpell, and Grammarly although the used LSTM and Transformer implementations are performing equality good. For the sequence accuracy good results are given by Google and BingSpell, the implementations of LSTM and Transformer are slightly worse than these tools but performing better than all others. The drop in the sequence accuracy for Grammarly is due to a comma analysis that is trying to solve comma errors, e.g. before conjunctions. Norvig is performing very poor in both categories which is due to the poor handling of capitalised words and numbers within the text, which results in the introduction of errors in every sentence. The bad sequence accuracy results for HunSpell and Aspell are due their limited punctuation

[9] http://norvig.com/spell-correct.html
[10] https://github.com/pytorch/fairseq

support and, as it seems, a limited word dictionary, where e.g. *iTunes* is corrected to *I Tunes*. For the neural network implementations LSTM and Transformer a drop for the sequence accuracy can be seen. This drop could be explained by the tendency of neural networks of being plagued by false positives when no actual error is present. The same behaviour can be seen for TextRazor and, as mentioned, Grammarly.

| | | *Acc. (in %)* | |
|---|---|---|---|
| **Tool** | **E** | **W** | **S** |
| Norvig | 63.61 | 63.61 | 0.00 |
| N-Gram | 81.94 | 81.94 | 7.88 |
| BingSpell | 99.56 | 99.53 | 97.10 |
| Google | 99.91 | 99.86 | 97.51 |
| Grammarly | 98.90 | 99.12 | 75.10 |
| TextRazor | 98.33 | 98.33 | 69.29 |
| LanguageTool | 94.29 | 96.77 | 68.05 |
| GrammarBot | 98.71 | 98.60 | 77.18 |
| PyEnchant | 94.80 | 94.29 | 28.22 |
| HunSpell | 84.38 | 83.93 | 06.64 |
| Aspell | 89.09 | 88.18 | 18.67 |
| JamSpell | 97.91 | 97.88 | 60.17 |
| LSTM | 99.72 | 99.73 | 93.36 |
| Transformer | 99.78 | 99.78 | 94.19 |

Table 1: Overview of the metrics $E_{Score}$ (E), word (W) and sequence accuracy (S) of all 14 spelling error correction tools when considering error-free sequences. Best scoring tools are highlighted in bold red.

**Accuracy**

The results for the word (**W**) and sequence accuracy (**S**) can be seen in Table 2. Although showing good results regarding the word accuracy, one may remember that $\approx 11\%/8\%$ of all tokens are error prone. Best word accuracy results are reported for BingSpell, Google, Grammarly, and the two neural network reference implementations LSTM and Transformer. The sequence accuracy represents the amount of sentences that were corrected so that it matches the groundtruth information. Here the best performing tools are BingSpell, Grammarly, LSTM, and Transformer. Although the tools BingSpell, Google, LanguageTool, and Grammarly also provide a confidence sorted list of alternative correction candidates there were several reasons we were not able to provide those here. On the one hand services like Grammarly and Google provided no API, just a web service, through which such information could be returned, on the other hand BingSpell had a very limited amount of free requests, which left us with using the account-free services. When taking a look at the results for $E_{Score}$ and $P_{Score}$ one may see that the results for $P_{Score}$ are always lower than the results of $E_{Score}$ meaning that all tools are falsely correcting NONE tokens. According to these scorings Grammarly, BingSpell, and the two neural network implementations perform best.

**Error Categories**

Table 3 gives an overview of the detection and correction F-Scores for each tool and error category and the number

| | | | | *Acc. (in %)* | |
|---|---|---|---|---|---|
| **Tool** | **E** | **P** | **SA** | **W** | **S** |
| Norvig | 17.06 | 8.12 | 0.45 | 57.08 | 0.00 |
| N-Gram | 12.49 | 5.05 | 0.66 | 73.17 | 0.25 |
| BingSpell | 46.98 | 41.89 | 0.93 | 92.17 | 20.55 |
| Google | 39.98 | 34.36 | 0.93 | 91.14 | 13.33 |
| Grammarly | 49.58 | 44.45 | 0.93 | 91.62 | 16.94 |
| TextRazor | 24.46 | 17.37 | 0.89 | 87.72 | 4.23 |
| LanguageTool | 41.20 | 34.64 | 0.89 | 87.80 | 9.22 |
| GrammarBot | 27.95 | 21.00 | 0.89 | 88.06 | 6.57 |
| PyEnchant | 23.07 | 15.61 | 0.84 | 84.54 | 3.55 |
| HunSpell | 21.11 | 12.88 | 0.70 | 75.51 | 0.50 |
| Aspell | 26.97 | 18.92 | 0.76 | 79.54 | 1.87 |
| JamSpell | 20.47 | 13.11 | 0.89 | 88.00 | 4.85 |
| LSTM | 55.11 | 50.77 | 0.93 | 93.00 | 22.67 |
| Transformer | 62.24 | 58.42 | 0.94 | 93.79 | 30.26 |

Table 2: Overview of the overall performance metrics $E_{Score}$ (E) and $P_{Score}$ (P), suggestion adequacy (SA), and word (W) and sequence accuracy (S) of all 14 spelling error correction tools. Best scoring tools are highlighted in bold, without marking the baseline methods.

of errors per category and test set in the first two rows. As mentioned, the greater the difference between $E_{Score}$ and $P_{Score}$ in Table 2 the lower the F-Score for NONE has to be, which can be seen in the first column. All tools are good at detecting and (not-)correcting tokens that are assigned as NONE, with BingSpell, Grammarly, and TextRazor providing the best results. Norvig and N-Gram are performing lower on that category, which could be due to the usage of a limited set of documents for Norvig and N-Gram and the preference for the more probable word. All tools are fairly good at detection and correction words assigned to NON_WORD, with BingSpell performing best. For the detection and correction of REAL_WORD BingSpell also provides the best result among all tools. Although SPLIT errors are detected by many tools only some tools are capable of correcting those errors, with Google performing best. HYPHENATION errors are corrected by almost any tool, with Google yielding the best results. CONCATENATION error get corrected well by BingSpell, interestingly followed by Aspell, which could be explained by the usage of a good word dictionary for Aspell. For CAPITALISATION one can see the trend of nearly all tools leaving the capitalisation of words as it is if the first letter is a capital one, only Grammarly is fixing some of these words beside the neural network approaches LSTM and Transformer. REPEAT errors are only handled by Grammarly, BingSpell, and LanguageTool, with the latter one performing best on that error type. All other tools are either not detecting any of those words as wrong (e.g. GrammarBot), or are detecting the repeated tokens but correcting them to other tokens (e.g. Aspell). LSTM and Transformer performing best, but it could also be the case that both networks have simply learned that two times the same word in a row should be merged to one. PUNCTUATION errors are, besides LSTM and Transformer, only handled by Grammarly, BingSpell, Google, and LanguageTool, with the first

| Tool | | NONE | NW | RW | SP | HY | CO | CA | RE | PU | MM | TE | CHY |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TinySet | | | 7,568 | 2,889 | 868 | 1,202 | 4,476 | 2,409 | 2,603 | 225 | 1,093 | 2,216 | 400 |
| SmallSet | | | 660 | 285 | 89 | 100 | 349 | 200 | 237 | 20 | 108 | 185 | 30 |
| Norvig | D | 0.72 | 0.44 | 0.08 | 0.77 | 0.68 | 0.08 | 0.13 | 0.35 | 0.01 | 0.30 | 0.03 | 0.00 |
| | C | 0.72 | 0.17 | 0.01 | 0.00 | 0.52 | 0.06 | 0.07 | 0.00 | 0.01 | 0.00 | 0.02 | 0.00 |
| N-Gram | D | 0.85 | 0.37 | 0.05 | 0.68 | 0.57 | 0.06 | 0.13 | 0.29 | 0.62 | 0.00 | 0.03 | 0.00 |
| | C | 0.85 | 0.03 | 0.01 | 0.00 | 0.13 | 0.01 | 0.05 | 0.00 | 0.47 | 0.00 | 0.02 | 0.00 |
| BingSpell | D | 0.97 | **0.77** | **0.37** | 0.91 | 0.62 | **0.93** | 0.15 | 0.93 | **0.71** | 0.07 | 0.18 | **0.00** |
| | C | 0.97 | **0.72** | **0.34** | 0.89 | 0.60 | **0.93** | 0.14 | **0.93** | **0.71** | 0.03 | 0.18 | **0.00** |
| Google | D | 0.96 | 0.75 | 0.31 | **0.94** | **0.78** | 0.91 | **0.01** | 0.03 | **0.71** | **0.00** | 0.13 | **0.00** |
| | C | 0.96 | 0.71 | 0.29 | **0.92** | **0.77** | 0.91 | **0.01** | 0.03 | **0.71** | **0.00** | 0.13 | **0.00** |
| Grammarly | D | 0.97 | 0.76 | 0.26 | 0.87 | 0.70 | 0.87 | **0.31** | 0.80 | **0.71** | **0.29** | **0.34** | 0.28 |
| | C | 0.97 | 0.71 | 0.23 | 0.83 | 0.68 | 0.87 | **0.31** | 0.80 | **0.71** | **0.23** | **0.34** | **0.28** |
| TextRazor | D | **0.98** | **0.56** | 0.15 | 0.70 | 0.75 | 0.16 | 0.03 | 0.07 | 0.67 | **0.00** | 0.09 | **0.35** |
| | C | **0.98** | **0.34** | 0.10 | **0.00** | 0.64 | 0.16 | **0.01** | **0.00** | 0.58 | **0.00** | 0.09 | **0.35** |
| LanguageTool | D | 0.95 | **0.56** | 0.24 | 0.85 | 0.71 | 0.87 | 0.23 | **0.94** | 0.61 | 0.07 | 0.13 | **0.00** |
| | C | 0.95 | 0.46 | 0.17 | 0.61 | 0.68 | 0.86 | 0.21 | **0.93** | 0.56 | **0.00** | 0.13 | **0.00** |
| GrammarBot | D | 0.95 | 0.62 | 0.26 | 0.78 | 0.64 | 0.72 | 0.20 | **0.01** | **0.71** | 0.04 | 0.13 | **0.00** |
| | C | 0.95 | 0.49 | 0.17 | **0.00** | 0.61 | 0.71 | 0.15 | **0.00** | 0.67 | **0.00** | 0.15 | **0.00** |
| PyEnchant | D | 0.93 | 0.58 | 0.18 | 0.80 | 0.28 | 0.83 | 0.16 | 0.10 | 0.68 | **0.00** | 0.03 | **0.00** |
| | C | 0.93 | 0.41 | 0.10 | **0.00** | 0.01 | 0.82 | 0.10 | **0.00** | 0.60 | **0.00** | 0.02 | **0.00** |
| HunSpell | D | **0.89** | 0.63 | **0.09** | 0.73 | **0.00** | 0.76 | 0.14 | 0.20 | 0.02 | **0.00** | **0.00** | **0.00** |
| | C | **0.89** | 0.46 | **0.09** | **0.00** | **0.00** | 0.76 | 0.08 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| Aspell | D | 0.91 | 0.57 | 0.12 | 0.76 | 0.76 | 0.91 | 0.10 | 0.26 | **0.00** | **0.00** | **0.00** | **0.00** |
| | C | 0.91 | 0.41 | 0.12 | **0.00** | 0.72 | 0.91 | 0.08 | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| JamSpell | D | 0.94 | 0.58 | 0.33 | **0.56** | **0.00** | **0.14** | 0.02 | 0.04 | 0.70 | 0.03 | 0.11 | **0.00** |
| | C | 0.94 | 0.50 | 0.27 | **0.00** | **0.00** | **0.14** | **0.01** | **0.00** | 0.66 | 0.03 | 0.10 | **0.00** |
| LSTM | D | **0.98** | 0.71 | 0.44 | 0.95 | 0.94 | 0.16 | 0.76 | 0.98 | 0.67 | 0.05 | 0.14 | 0.21 |
| | C | **0.98** | 0.64 | 0.37 | 0.87 | 0.93 | 0.16 | 0.76 | 0.98 | 0.67 | 0.00 | 0.14 | 0.21 |
| Transformer | D | **0.98** | **0.75** | **0.58** | **0.97** | **0.96** | **0.23** | **0.83** | **1.00** | **0.71** | **0.20** | **0.28** | **0.22** |
| | C | **0.98** | **0.69** | **0.54** | **0.95** | **0.95** | **0.23** | **0.82** | **1.00** | **0.71** | **0.13** | **0.28** | **0.22** |

Table 3: Detailed information about the *F-Score* performance of each of the 14 state-of-the-art spelling error correction tools and the two baselines. For all tools and all previously described error categories the *F-Score* of the detection and correction is listed. NONE: Words without error; NW: NON_WORD errors; RW: REAL_WORD errors; SP: SPLIT errors; HY: HYPHENATION errors; CA: CAPITALISATION errors; RE: REPEAT errors; MM: MENTION_MISMATCH errors; TE: TENSE errors; CO: CONCATENATION errors; CHY: COMPOUND_HYPHEN errors. **D** represents the Detection, **C** represents the Correction. Best scoring tools are highlighted in bold, without marking the baseline methods.

three ones performing equality good. All other tools seem not to be capable of indicating wrong usages of brackets and other punctuation errors like wrong comma usage. The same holds for MENTION_MISMATCH errors, where only Grammarly is capable of detecting and correcting at least some errors. Interestingly, although the used training set for LSTM and Transformer does not contain any error of that category, Transformer is also capable of detecting and correcting some of the occuring errors. For the error category TENSE only BingSpell, Google, and Grammarly were able to fix at least some errors, with Grammarly performing best. The generated training set for LSTM and Transformer does not contain TENSE errors, but both implementations were able to detect and fix some of those errors. The error category COMPOUND_HYPHEN can only be detected by Grammarly and TextRazor, with TextRazor performing best. The last three error categories can also be seen as the hardest to find and correct because they require not only an understanding of the current and previous sentences but also knowledge about mentioned entities.

## 6. Conclusion

We presented the generation of a test set containing sentences from Wikipedia, distorted using realistic error models. We further presented an algorithm through which we can build the alignments between the erroneous source, groundtruth, and proposed correction. With these alignments we evaluated 14 tools, based on the generated test sets, and collected metrics to measure their detection and correction performance on several error categories. As seen in the results and discussion, all tools are good at correcting non-words, but error categories that are difficult to detect like real-word, mention-mismatch, or tense errors require knowledge about the sentence and occuring entities. They further need a conceptional representation and understanding of the whole document for finding long dependency errors like mention mismatches. Fetching errors that are based on inadequate knowledge would further need the introduction of knowledge bases and knowledge graphs, making it possible to add additional information about identified entities to the correction of the sentence.

# 7. References

## 7.1. Bibliographical References


Bast, H. and Haussmann, E. (2015). More accurate question answering on freebase. In Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015, pages 1431–1440.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In 38th Annual Meeting of the Association for Computational Linguistics, Hong Kong, China, October 1-8, 2000.

Ghosh, S. and Kristensson, P. O. (2017). Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429.

Google. (2019). Google. https://www.google.com/drive.

GrammarBot. (2019). GrammarBot. http://grammarbot.io/.

Grammarly Inc. (2019). Grammarly. https://www.grammarly.com.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Kevin Atkinson. (2017). Aspell. http://aspell.net/.

LanguageTool.org. (2019). LanguageTool. https://languagetool.org/.

Li, H., Wang, Y., Liu, X., Sheng, Z., and Wei, S. (2018). Spelling error correction using a nested RNN model and pseudo training data. *CoRR*, abs/1811.00238.

Liang, F. M. (1983). Word hy-phen-a-tion by com-puter. Technical report, Calif. Univ. Stanford. Comput. Sci. Dept.

Lichtarge, J., Alberti, C., Kumar, S., Shazeer, N., Parmar, N., and Tong, S. (2019). Corpora generation for grammatical error correction. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 3291–3301.

László Németh. (2019). Hunspell. http://hunspell.github.io/.

Microsoft Corp. (2019). BingSpell. https://azure.microsoft.com/de-de/services/cognitive-services/spell-check/.

Ozinov, Filipp. (2019). JamSpell. https://github.com/bakwc/JamSpell.

Philips, L. (2000). The double metaphone search algorithm. *C/C++ users journal*, 18(6):38–43.

Ryan Kelly. (2019). PyEnchant. https://github.com/rfk/pyenchant.

Starlander, M. and Popescu-Belis, A. (2002). Corpus-based evaluation of a french spelling and grammar checker. In Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain.

Tetreault, J. R., Sakaguchi, K., and Napoles, C. (2017). JFLEG: A fluency corpus and benchmark for grammatical error correction. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers, pages 229–234.

textrazor.com. (2019). TextRazor. https://www.textrazor.com.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pages 6000–6010.