Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Studiengang Informatik, B.Sc.

Query Spelling Correction using Pre-trained Word Embeddings

Bachelorarbeit

Gustav Lahmann

Betreuer: Prof. Dr. Matthias Hagen,
Ines Zelch

Jena, den 27. April 2024

Zusammenfassung

Ein Großteil der Interaktion mit Webdiensten findet über Suchmasken statt. Wenn Nutzer eine Suchanfrage an Websuchmaschinen stellen, erwarten sie korrekte Ergebnisse, auch wenn die formulierte Anfrage Schreibfehler enthält. Nicht alle Retrievalsysteme können von sich aus mit falsch geschriebenen Wörtern umgehen. Daher ist es notwendig, die Fehler in der Suchanfrage zu korrigieren, bevor diese weiterverarbeitet wird.

In dieser Arbeit testen wir die etablierte Rechtschreibkorrektur für Textdokumente „Hunspell“, sowie den in vielen Benchmarks als Baseline verwendeten „Pyspellchecker“ auf einem annotierten Datensatz echter Suchanfragen aus dem Webis QSpell Korpus. Wir stellen fest, dass beide Verfahren nicht für die automatische Korrektur von Suchanfragen geeignet sind, da Eigennamen und Abkürzungen zu streng, und daher in sinnfremde Wörter, korriegiert werden.

Um eine für Suchanfragen geeignete Rechtschreibkorrektur zu entwickeln, untersuchen wir ein von Ed Rushton vorgeschlagenes Verfahren unter der Verwendung von vortrainierten GloVe Wortvektoren. Durch eine Transformation der Wortvektoren im Vektorraum kann zu einer in den GloVe Vektoren enthaltenen Falschschreibung eines Wortes die korrekt geschriebene Form gefunden werden. Das Ergebnis ist eine Substitutionstabelle von häufigen Rechtschreibfehlern und ihren Korrekturen. Wird diese kontextfrei auf die Wörter der Suchanfragen des Webis QSpell Datensatzes angewendet, werden dadurch mehr Anfragen korrigiert, als dass Anfragen sinnentstellt werden. Zum Vergleich wurden Korrekturen der Google Suche sowie der Bing Spell Check API abgefragt, welche eine Verbesserung beider Dienste im Vergleich zu 2017 aufzeigen. Die erhaltenen Korrekturen für den QSpell Datensatz können für zukünftige Projekte genutzt werden und werden mit dem Code für diese Arbeit veröffentlicht.

Contents

1 Introduction

The main way we interact with web search engines, online shops, or music streaming catalogues today is through natural language search queries. The users expect the retrieval system to understand their intent, even if the query is not spelled correctly. In this thesis, we evaluate the established spell checker for text documents Hunspell as well as the commonly used baseline Pyspellchecker on real-world search engine queries from the Webis QSpell corpus [1]. For comparison, we use newly crawled spelling suggestions from Google Search and the Bing Spell Check API, as well as the suggestions Bing and Google returned when crawled by Hagen et al. in 2017. We find both Hunspell and Pyspellchecker to be unsuitable for correcting web search queries, as they correct abbreviations and entity names too strictly and thereby corrupt originally correct queries.

In an attempt to build a spell checker specifically for search queries, we explore an idea of Ed Rushton to use GloVe word embeddings for detecting and correcting spelling mistakes [7]. Word embeddings are low-dimensional, dense vector representations for words, that are learned from unlabeled natural language corpora in a way that the word embeddings of semantically similar words are close to each other in the vector space. Each dimension of the vector space then encodes syntactic or semantic properties of the word, which allows for extracting relations between two words and adding them to a third word. For example, adding the difference between the vectors for “cars” and “car” to the word vector of “apple” results in a vector which is closest to the word embedding for “apples”, as it represents the “plural” relationship [4]. Rushton found that GloVe vectors encode such a relationship between misspellings of a word and their correctly spelled counterparts. Starting with a list of commonly misspelled words and their correct forms, he calculated a spell

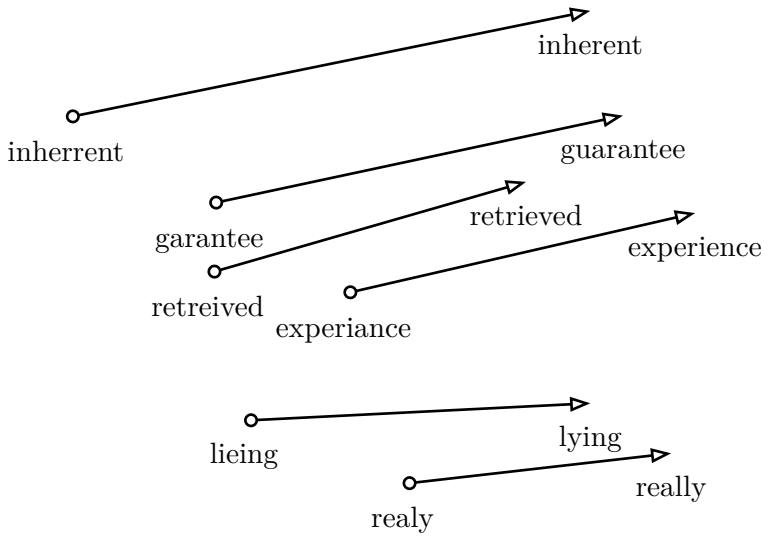

Figure 1. Difference vectors between word embeddings of misspelled words and their correct spellings projected into 2D space using a PCA.

correction vector, that ends up close to the embedding of the correction when added to the embedding of an incorrect word.

We reimplement Rushton's approach and discuss the intuition and effectiveness of its steps. The resulting spell checker is a context-insensitive lookup table of (*correct, incorrect*) pairs extracted from the GloVe word embeddings, which, applied to the Webis QSpell dataset, outperforms the do-nothing baseline, but lacks behind the spell corrector used in Google or Bing.

2 Background

In the broadest sense, a misspelling of a query occurrs when the actual input query passed to the retrieval system deviates from the query the user formed in their mind. This could be due to hitting an adjacent key on a keyboard, typing a similar sounding word because of lacking knowledge in normative spelling, or an autocorrection system on a mobile phone guessing a different word from what the user intended to write.

For detecting misspelled words, spelling mistakes fall in two categories. Typing a wrong letter will usually result in a so called *non-word error*, meaning the misspelled word does not exist. These mistakes are traditionally detected by checking against a dictionary of the target language. Choosing the right size of this dictionary is crucial for the effectiveness of this approach. When restricting the list of valid words to curated dictionaries, terms commonly found in search queries like proper names, product descriptions and newly emerging internet slang cannot be kept up with. In contrast, constructing a list of valid words from the most commonly used words in a document collection will introduce common misspellings present in the documents to the dictionary, rendering it useless.

The second case of misspellings are *real-word errors*. Here, the misspelled word itself is a valid, yet semantically different word, which occurrs in the language's dictionary. Therefore, real-word errors can only be found to be incorrect in the context of the surrounding words of the query. Further cases of typos especially relevant in search queries are wrongly added or omitted spaces. As spaces are usually used for splitting a query into words, these errors have to be corrected across and within word boundaries, making methods operating on single words ineffective.

In this thesis, we treat query spelling correction as the second part of the query preprocessing pipeline, right after tokenizing a query into words at every whitespace. Spell correction is applied before passing the modified query to the retrieval system, preventing the retrieved document collection to affect the decision of the spell corrector. While this makes the approaches applicable to various retrieval models, the robustness against misspellings that the retrieval system possesses on its own is important to keep in mind when designing the spell corrector.

3 Related Works

In recent years, query spelling correction has been mainly explored in combination with dense retrievers, a new retrieval system used for passage retrieval. However, algorithms for spelling correction reach back to way before computers were used for text processing. For example, the Soundex algorithm was developed around 1918 and is still implemented in all popular database servers as of today. The variety of spell checking approaches developed over the last century is why we restrict this section to explaining the algorithms of Hunspell and Pyspellchecker in depth, instead of trying to provide an overview over spell checking algorithms in general. In the last section we describe the GloVe language model, which is the model used by Rushton's spell checking approach.

3.1 Robustness in Dense Retrievers

The effect of misspellings on retrieval effectiveness largely depends on the underlying retrieval system. Established sparse retrieval models like tf-idf and BM25 already suffer from vocabulary mismatch, which means that synonym words are used to describe an information need with terms that cannot be found in the relevant documents. Obviously, these approaches are not robust against spelling mistakes.

To overcome the problem of vocabulary mismatch and retrieve documents about a certain topic instead of a specific choice of words, *dense retrievers* gained popularity since 2020. Based on a pretrained BERT model, dense vector representations for queries and documents are learned in a way that the dot product of query and document vector is maximized for relevant query-document pairs. At interference time, the vector representation of the query is calculated and relevant documents are retrieved by finding the nearest neighbors in a precomputed index of document vectors [2]. The query vectors for synonym queries are close to each other and hence yield the same results, solving the vocabulary mismatch problem. However, Zhuang and Zuccon found that classical dense retrievers are not resistant to spelling mistakes [11]. Passage retrieval datasets used for training dense retrievers, such as MS MARCO, do not contain more than a handful of spelling mistakes, as the dataset was manually curated. Queries containing typos are therefore out-of-distribution and result in worse retrieval effectiveness. To overcome this weakness, a BERT dense retriever was trained on a dataset augmented with artificial spelling mistakes. The typos were injected into the original MS MARCO dataset by introducing random character insertions, deletions, substitutions and swaps, as well as swaps of characters that are adjacent on the QUERTY keyboard layout. The dense retriever trained with this dataset was more effective on queries containing typos than the standard system, while keeping the same scores on queries without spelling mistakes.

In a following paper, Zhuang and Zuccon explored the underlying reason why BERT based models are affected by typos. The WordPiece tokenizer used by BERT assigns a single token to commonly used words, while less commonly used words are built out of multiple tokens, each representing a part of the larger word [12]. When introducing a single character spelling mistake, a word that was tokenized as a single token before, might now be tokenized as multiple tokens because the misspelled word is too uncommon to have its own token. The query encoded as tokens and passed to the dense retrieval model therefore looks completely different,

even if the mistake only affected one single character. To make the query encoding robust against spelling mistakes, Zhuang and Zuccon replaced the WordPiece tokenizer by a CharacterCNN, which encodes similar words into vectors that are each other. The resulting dense retriever was better than a normal BERT dense retriever, where queries were first corrected using Pyspellchecker. However, when used in combination with Microsoft's Bing Spell Check API, the normal model still outperformed the CharacterCNN based robust dense retriever.

In this thesis, we follow Zhuang and Zuccon by evaluating the same baseline spellcheckers, Pyspellchecker and Bing. However, instead of using artificially introduced spelling mistakes, we use the real-world mistakes found in the QSpell dataset.

3.2 Norvig's Spell Checker

The spell correction implemented in Pyspellchecker[1] is based on Peter Norvig's article "How to Write a Spelling Corrector", that was spontaneously written on an intercontinental flight in 2007 [5]. Norvig's spell checker is a simplified version of the noisy channel model, first described in 1990 by Kernighan et al. [3].

The noisy channel model assumes the word, which the user intends to write, passes through a noisy channel before entering the system, which changes the word to a possibly misspelled word w. The source for the added noise can be sloppy typing or bad orthography by the user. In order to recover the originally intended word from the possibly misspelled word w, a list of correction candidates is selected from a dictionary. This is done by finding words in the dictionary that have a so called *Levenshtein distance* of one, which means the word can be obtained from w by inserting, deleting, swapping or substituting one character.

Out of all candidates with a Levenshtein distance of one, the candidate maximizing the probability $P(c|w)$ is chosen, that is the probability that correction candidate c is the intended word, given the observed word w leaving the noisy channel.

Using Bayes' Theorem, it follows that $P(c|w) = P(c)\frac{P(w|c)}{P(w)}$, in which $P(w)$ is independent of the candidate c and therefore irrelevant when looking for the maximum probability over all the candidates. The correction candidate can be selected out of a set of possible candidates by calculating

$$\text{argmax}_{c \in \text{candidates}}\ P(c)P(w|c).$$

The probability $P(c)$ is called the *language model*, as it describes the probability of observing the word c in a text. It can be obtained by counting the occurrences of c in a text corpus. The conditional probability $P(w|c)$ is the *error model*, describing the probability that the word c was mistakenly turned into w by the noisy channel. Kernighan et al. calculated confusion matrices for each kind of the above-mentioned error types with edit distance of one, from a list of misspelled words [3]. The four matrices show how often the character x is deleted and inserted after character y, how often characters x and y are swapped, and the number of times x is substituted by y. Calculating $P(w|c)$ is a matter of detecting the error type, looking up the occurrences of that error in the corresponding matrix, and lastly dividing the

[1] https://github.com/barrust/pyspellchecker

occurrences obtained from the matrix by the occurrences of the involved characters in the training list of misspellings.

Since Norvig did not have Internet access during the flight, he resorted to a much simpler error model. He omitted the $P(w|c)$ factor and instead chose the correction candidates for w in the following way: If w is in the dictionary, it is chosen. Otherwise, the candidate c with the larges $P(c)$ is chosen out of all candidates in the dictionary with Levenshtein distance one. If there are no candidates with Levenshtein distance one, candidates with Levenshtein distance two are used instead (these are words that can be obtained from w by applying two of the errors explained above). If the list with an edit distance of two is empty as well, the unknown word w is returned, assuming it is a correctly spelled unknown entity name.

Norvig's spell checker corrected 68% of the test split of the Birkbeck spelling error corpus, while Kernighan et al. reported 87% accuracy on their manually judged spelling mistakes found in the 1988 AP corpus. The Pyspellchecker implementation used by Zhuang and Zuccon as a baseline uses a dictionary obtained by applying cleaning rules to English subtitles from the OpenSubtitles corpus.

3.3 **Hunspell**

Hunspell is one of the most popular open source spell checkers, being used by Mozilla Firefox and Thunderbird, Google Chrome, LibreOffice, and Adobe Illustrator and InDesign, only to name the most well-known programs. Originally developed for Hungarian, it now supports more than 130 languages when provided with the language specific `.dic` and `.aff` files. As the spell checking algorithm was expanded over time, its official documentation is lacking. Fortunately, in 2021 Victor Shepelev reimplemented the core parts of Hunspell in Python in his project *Spylls* and documented the inner workings of Hunspell in a series of blog posts [8].

As the name suggests, the `.dic` file contains a dictionary of words of the language. However, only the stems of words are stored in it, followed by a list of flags for each stem. The flags can be found in the affix `.aff` file, where their meaning is defined. The `SFX` and `PFX` keywords define a flag to represent a certain suffix or prefix. For a single flag, different rules can match on the stem, and depending on parts of it, allow for certain suffixes or prefixes. Another kind of flags are `COMPOUNDFLAG`s, which signal whether words can be parts of compounds, and in which positions. There are even `COMPOUNDRULES` which specify regular expressions of how stems with certain compound flags can be combined. This for example allows defining the rules for arbitrary English numerals (1st, 2nd, …, 901st).

These were just a few of the flag types that can be specified in the affix file. Their purpose is to reduce the size of the dictionary file, while still covering all the possible forms words can occur in. This is especially important in languages other than English, which use compounds and prefixes heavily. The compression algorithm comes at the cost of fast lookup times. In order to find out whether a word is present in the dictionary, the correct stem has to be identified, and depending on the flags stored with the stem, the allowed compounds and affixes have to be considered. To limit the runtime of this lookup, Hunspell allows for at maximum two prefixes and suffixes per word, which is not enough for some languages, which will then have to include affixes in the stems themselves, resulting in a larger dictionary file.

Hunspell is an entirely context free spell checker. In the *lookup* stage, each word of the text is looked up in the dictionary, while following the rules specified in the affix file. When no matching word can be found, the *suggest* phase will generate a list of candidate words for the user to select from. This is done in two steps, an edit-based search and a full dictionary search, where the latter is only employed in cases the first one yielded no results.

The edit-based search follows a similar approach to the one described by Norvig, but is much more advanced in the edits that are applied to the misspelled word. Additionally, the transformations on the word are influenced by rules given in the `.aff` file, for example the `REP` table can define replacements (like "f" to "ph"), and the `MAP` table defines similar characters (like "a", "å" and "ä"). Further transformations include uppercasing of the whole word, inserting spaces or dashes at every position, swapping two adjacent letters, swapping non-adjacent letters up to 4 characters apart, replacing letters by their neighbors on the keyboard (whose layout is defined in the `KEY` section of an `.aff` file), and many more variants of insertions, deletions and splitting. After each of these transformations, whenever the resulting word occurs in the dictionary, it is returned as a correction candidate. Therefore, the order of the candidates is given by the order in which the transformations are tried out. If none of the transformation rules results in a valid word from the dictionary, a full dictionary search is performed as the second phase.

Because of the possible affixes for each stem, iterating over the whole list of allowed words is expensive. Therefore, when performing a full dictionary search at first only stems are used for comparing with. Only for the top 100 most similar stems to the misspelled word, all forms allowed in the affix file are generated. Out of these the top 200 most similar words to the misspelled word are selected. Only then a detailed similarity score is calculated, based on features like n-gram counting, the longest common substring length, the leftmost common substring length, the number of equal characters at the same positions and a weighted n-gram score. The n-gram count of two strings is the number of common substrings of length 1 to n and in a weighted n-gram score the absence of common substrings adds negative points. All of these features are weighted, with some of the weights being tunable in the `.aff` file.

In his blog post, Shepelev summarizes the rules of the scoring function the following way:

> It seems that they were grown through years of trial-and-experimentation to choose the coefficients and the order of summands – not related to any theoretical linguistic meaning but just found empirically. Those parts of the algorithms strongly resemble how machine learning works; only, in this case, it were Hunspell developers who adjusted "weights" after each evaluation.
>
> — Victor Shepelev

At last, Hunspell supports searching for correction candidates based on similar pronunciation. When provided with a `PHONE` table in the affix file, which follows the same format as the older, open source spell checker Aspell, misspelled words can

be converted into a representation based on the phonetics of the word. Hunspell then searches through the phonetic representations generated for all the stems in the dictionary and returns correction candidates that have the same representation, and hence a similar pronunciation. This feature is pretty much unused in practice, since out of all dictionaries distributed with Firefox and LibreOffice only the `en_ZA` dictionary includes a `PHONE` table. Additionally, Hunspell only searches through the stems, so words containing affixes or compounds cannot be found.

Due to its wide adoption and huge flexibility of language specific customization in the affix file, Hunspell has worked out the specific quirks of many languages, justifying its position as the standard spell checker.

3.4 GloVe Word Embeddings

In analogy to how dense retrievers learn to encode the semantics of queries and documents as dense vector representations, *word embeddings* are vector representations for single words. The goal is to represent semantic relationships between words as relationships between the vector representations. Words that have a similar semantic meaning should have word embeddings that are close to each other in the vector space.

The meaning of a word can be inferred by the words that appear in its context. For example, you can easily guess the omitted ____ just by having read the surrounding sentence. Maybe it is unclear whether the original sentence said "word", "term", or "phrase", but all of these have a similar meaning that makes sense in the context. Given a large corpus of natural language, each word appears in many different contexts, which define its relations to other words. Capturing these relations as a vector is the objective of learning word embeddings from a text corpus.

Google's *word2vec* library published in 2013 implemented two different approaches to learn word embeddings from text. The *skip-gram* model tries to predict the context of a word, given the word itself, while the *CBOW* (continuous bag-of-words) model predicts the missing word given its context, similar to the example of the previous paragraph. During training, both methods iterate over context windows of the corpus and adjust the vector representations to allow for better predictions as they process more of the text.

Instead of looking at windows of text at a time, the *GloVe* (Global Vectors) model [6], developed by Pennington et al. at Stanford in 2014, first creates a statistic of the whole corpus, which is only then used to learn word embeddings. Taking inspiration from term-document matrices used in LSA (latent semantic analysis), a term-term co-occurrence matrix X is constructed over a corpus. For all words i and j found in the vocabulary of the corpus, X_{ij} stores how often j occurred in the context window of i. The exact value is accumulated by sliding the context window over the whole corpus, where the ten words before and ten words after a word are seen as the context of the word. A word in the context window which is d words apart from the current word contributes $\frac{1}{d}$ to the count of the current word, accounting for the diminishing importance of a word with increasing distance.

After calculating the term-term co-occurrence matrix X, the probability of the word k occurring in the context of i is given by

$$P_{ik} = \frac{X_{ik}}{\sum_l X_{il}}$$

This probability, however, doesn't say much about the relation between the words k and i, since it is unknown whether the word k is for example unlikely to appear in the context of i because it is unrelated, or because it is simply an uncommon, but related word. For example, the word "water" is much more probable to appear in the context of "ice" than is the word "solid". That is not because "ice" is not related to "solid", but rather because "water" is a word used more often than "solid" [6].

To determine the relationship $k = \text{solid}$ has to $i = \text{ice}$, it would be interesting to know, how probable "solid" is in another context, for example in the context of $j = \text{steam}$. By looking at the ratio of probabilities $\frac{P_{ik}}{P_{jk}}$, the likeliness of k appearing in any context just by being a common word cancels out. In this example, the ratio will be high, because "solid" is more probable in the context of "ice" than it is in the context of "steam". In analogy, for $k = \text{gas}$ the ratio would be low, as "gas" is less likely in the context of "ice" than it is in the context of "steam". If k is an unrelated word to both "ice" and "steam", for example "fashion", or a word related to both of them in the same way, such as "water", the ratio would be close to 1. Each probability would not depend on the context, as the context is unrelated, so both should be about the same.

Based on this observation about the relationship being captured by this ratio of probabilities, the GloVe model aims to learn word vectors for the words i, j and k in a way that some function F can calculate $F\big(w_i, w_j, \tilde{w}_k\big) = \frac{P_{ik}}{P_{jk}}$ based on the vectors w_i, w_j and $\tilde{w}_k$. The possible options for F are then reduced by putting specific constraints on F. The function should only depend on the difference between w_i and w_j, be linear in its arguments, be invariant to transpositions of the co-occurrence matrix and be a homomorphism between the group of rational numbers regarding addition and the group of positive rational numbers regarding multiplication. Pennington et al. end up with the equation $w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$, in which the bias terms b_i and $\tilde{b}_k$ are introduced to make it symmetric. After adding a weighting function dependent on X_{ij} to limit the impact of very common words and avoid taking the logarithm of zero for words that do not appear in the context, the equation can be formulated as a least squares regression problem.

The GloVe word embeddings were trained on multiple dumps of Wikipedia, as well as on a Common Crawl corpus from 2014. Vector dimensions larger than 300 were empirically found to provide diminishing returns on semantic and syntactic word analogy tasks. Overall, GloVe word vectors outperformed word2vec models in named entity recognition, word analogy and word similarity tasks while also having a lower required training time.

4 Spelling Correction using GloVe Word Embeddings

In the GloVe paper, Pennington et al. describe how to use the linear structures in the word embeddings to solve the word analogy task. To answer questions such as "Athens is to Greece as Berlin is to _____" they calculate $w_{\text{Greece}} - w_{\text{Athens}} + w_{\text{Berlin}}$. The word whose GloVe embedding is closest to the calculated vector by cosine similarity is the answer to the analogy question. Pennington et al. show that the GloVe model successfully captures semantic relationships between words, such as *male – female*, *company – CEO*, and *city – zip code*, but also syntactic properties like *adjective – comparative*.

The idea Ed Rushton explored in a blog post in 2018 was that the misspelling of words could be a further syntactic relationship that was automatically learned during training of the GloVe vectors [7]. Finding the correct spelling of a misspelled word would be reduced to a word analogy task of the form "lieing is to lying as annouced is to _____", yielding the correctly spelled "announced" when calculating $w_{\text{lying}} - w_{\text{lieing}} + w_{\text{annouced}}$ and finding the word vector with the highest cosine similarity to the calculated vector.

The cosine similarity between two vectors $x, y \in \mathbb{R}^n$ is defined as

$$\cos(x, y) = \frac{\langle x, y \rangle}{\|x\|\|y\|}.$$

To find the vector x_{nearest} out of all GloVe vectors that maximizes the cosine similarity to a given vector y, nearest neighbor search libraries can be used to achieve optimal performance. We use the library `faiss`, which is developed by Facebook, and was also used by Rushton. The *inner product* metric offered by `faiss` corresponds to the cosine similarity between normalized vectors because if $\|x\| = \|y\| = 1$, then $\cos(x, y) = \langle x, y \rangle$. That is why all GloVe vectors are normalized before adding them to the index to search in. In analogy, the vector to search for has to be normalized before using `faiss` to find the nearest neighbors in the index. Conceptually, normalizing the vectors places them on a hypersphere around the origin with radius one. Searching for the maximum inner product (so maximum cosine similarity) between these normalized vectors is equivalent to finding the vector with minimum euclidean distance[2]. This can be seen by writing the squared euclidean distance between two normalized vectors x and y as

$$\|x - y\|^2 = \langle x - y, x - y \rangle = \langle x, x \rangle - 2\langle x, y \rangle + \langle y, y \rangle = 2 - 2\langle x, y \rangle$$

4.1 Finding a Spell Correction Vector

Following the assumption that there is a vector representing the "is correct spelling of" relationship between a correct and a misspelled word, Rushton tried to find this vector by averaging across multiple (*correct, incorrect*) pairs. To do so, he started with a list of 111 common misspellings from the Oxford English Dictionary and trained a spell correction vector using them. When taking the average over the difference vectors between the (*correct, incorrect*) pairs, the vectors that are averaged should be normalized. Otherwise vectors with greater length would have a stronger

[2] https://github.com/facebookresearch/faiss/wiki/MetricType-and-distances

weight in determining the average direction. Diverging from Rushtons implementation, these intuitions lead to the following formula

$$v_{\text{avg}} = \frac{1}{|\text{pairs}|} \sum_{(c,i)\in \text{ pairs}} \frac{w_c - w_i}{\|w_c - w_i\|}$$

from which the final spelling correction vector is obtained by normalizing it, since we only care about its direction and will scale it suitably later on:

$$v_{\text{sc}} = \frac{v_{\text{avg}}}{\|v_{\text{avg}}\|}$$

In contrast to this intuition, Rushton always works with the normalized embedding vectors, so he calculates

$$v_{\text{avg rushton}} = \frac{1}{|\text{pairs}|} \sum_{(c,i)\in \text{ pairs}} \frac{w_c}{\|w_c\|} - \frac{w_i}{\|w_i\|}$$

which corresponds to determining the direction between the vectors lying on the radius-one hypersphere instead of the actual GloVe embeddings. We found both approaches to result in similar spelling correction vectors with equal correction effectiveness. Because the explanation is more intuitive, we chose to use the first approach going forward.

After calculating the initial spelling correction vector, Rushton observes the cosine similarity between v_{sc} and a normalized word embedding w_{word} is positive for correctly spelled words, whereas misspelled words lead to a negative similarity score, as visualized in Figure 2. He then samples some probably correctly spelled words from the GloVe corpus by selecting those with a large, positive cosine similarity to v_{sc}. By subtracting v_{sc} from the correctly spelled word and performing a nearest

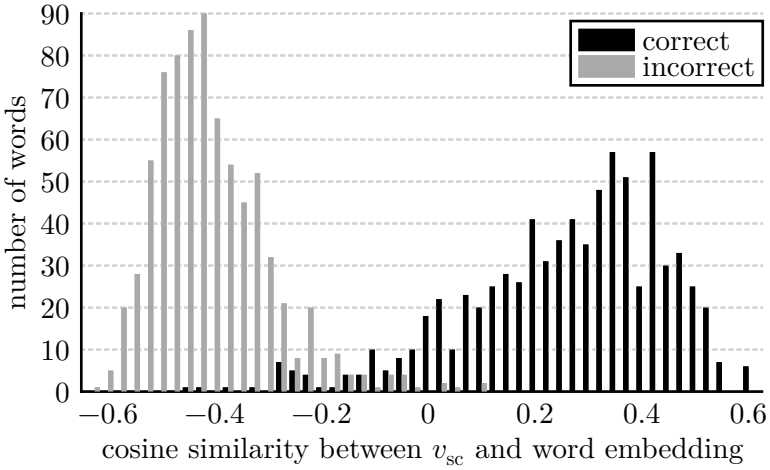

FIGURE 2. Correct and incorrect words from the Wikipedia list of common misspellings as a histogram of their cosine similarity to the spell correction vector, trained on the same list of (*correct, incorrect*) pairs. Correctly spelled words have a positive cosine similarity, whereas misspelled words have a negative similarity score.

neighbor search, Rushton builds a larger list of (*correct, incorrect*) pairs, from which he then trains a second spelling correction vector, that generalizes better than the first one. Instead of following this two-step process, we chose to simply use a larger existing list of word pairs, specifically the one obtained from the Wikipedia list of 5,712 common misspellings[3] from A to Z. As can be seen in the final evaluation in Section 5.3.3, the spelling correction vector trained on this list performs slightly better than using Rushton's correction vector.

4.2 Correcting a Misspelled Word

The general procedure for spell checking a single word as proposed by Rushton consists of two steps. At first, v_{sc} is added to the embedding w_{word} of the possibly misspelled word. A nearest neighbor search yields the top five GloVe vectors with the highest cosine similarity to $v_{\text{sc}} + w_{\text{word}}$. In the next step, the five correction candidates are iterated in order of increasing distance. If a candidate word is equal to the possibly misspelled word, the word is considered to be spelled correctly and is returned. Otherwise the `is_misspelling` function, developed by Rushton, described in Listing 1 is evaluated for the candidate word and the supposedly misspelled word.

```
is_misspelling(w1, w2):
    w1 = w1.lower()
    w2 = w2.lower()
    if w1 == w2:                                          # capitalization
        return False
    if (w1[−1] == "s" and w2[−1] ≠ "s") or (w1[−1] ≠ "s"
    and w2[−1] == "s"):                                   # plural
        return False
    if w1[: −2] == w2 or w2[: −2] == w1:                  # suffix
        return False
    if w1[2 :] == w2 or w2[2 :] == w1:                    # prefix
        return False
    if (w1[−1] == "e" and w2[−1] == "d") or (w1[−1] ==
    "d" and w2[−1] == "e"):                               # added d
        return False
    return lev(w1, w2) ≤ 2
```

LISTING 1. Rushton's function determines whether w_2 is a plausible misspelling of w_1 by rejecting equal words of different capitalization, plural forms, words with pre- or suffixes and forms with an added "d". Accepted misspellings are required to have a Levenshtein distance less or equal two.

[3] https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings

If the function determines the candidate word to be a misspelling, the correction candidate is returned and the iteration stops. If the function evaluates to false, the next candidate is checked. If none of the five correction candidates satisfy the `is_misspelling` function, the original word is considered to be correct again and will be returned.

4.3 **How far to correct**

When Rushton evaluated the spell correction method described above, he noticed the correct correction candidate sometimes does not make it into the five nearest neighbors. His intuition was that some words require a stronger correction than others, meaning the spell correction vector has to be scaled to end up close to the correction candidate. Rushton found the amount of scaling depends on how misspelled the word is, which is indicated by the cosine similarity to v_{sc}. Rushton further assumed the relation between the cosine similarity and the factor k, by which to scale v_{sc}, can be modelled using a linear function

$$k = m \cos\left(v_{\text{sc}}, \frac{w_{\text{word}}}{\|w_{\text{word}}\|}\right) + n$$

which we will analyze empirically in the following section.

For the set of (*correct, incorrect*) pairs from the Wikipedia list of common misspellings, we can calculate how much the normalized spell correction vector v_{sc} has to be scaled after being added to the incorrect word vector $w_{\text{incorrect}}$, to end up closest to the word embedding w_{correct} of the correct word. The vector $w_{\text{incorrect}} + k \cdot v_{\text{sc}}$ is closest to w_{correct} when the difference vector $w_{\text{correct}} - (w_{\text{incorrect}} + k \cdot v_{\text{sc}})$ is perpendicular to v_{sc}, as can be seen in Figure 3.

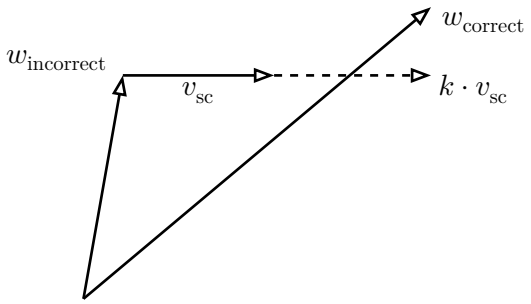

FIGURE 3. For a given pair of a correct word with embedding w_{correct} and a misspelling with embedding $w_{\text{incorrect}}$, the factor k, by which to scale v_{sc} by, can be calculated by choosing k in such a way that the remaining difference vector to w_{correct} is perpendicular to v_{sc}.

By demanding the inner product between the vectors to be zero, the formula for the optimal k to scale v_{sc}, can be obtained. It can be further simplified in the last step because v_{sc} is normalized, so the inner product $\langle v_{\text{sc}}, v_{\text{sc}} \rangle$ is one.

$$\begin{aligned} 0 &= \langle w_{\text{correct}} - (w_{\text{incorrect}} + k \cdot v_{\text{sc}}), v_{\text{sc}} \rangle \\ \iff k &= \frac{\langle w_{\text{correct}} - w_{\text{incorrect}}, v_{\text{sc}} \rangle}{\langle v_{\text{sc}}, v_{\text{sc}} \rangle} \\ &= \langle w_{\text{correct}} - w_{\text{incorrect}}, v_{\text{sc}} \rangle \end{aligned}$$

For investigating the relationship between the optimal scaling factor k and the cosine similarity between v_{sc} and the word embedding of misspelled words, Figure 4 plots (*correct, incorrect*) pairs from the Wikipedia list of common misspellings. By running a linear regression, the parameters m and n for the linear fit are computed. They are visualized as a black line in Figure 4, whereas the parameters obtained by Rushton using an optimization algorithm are represented by the gray line.

Although the linear fit looks loosely like it would match the data points, using the scaling factor k as determined by the calculated parameters m and n does not result in a better correction accuracy. To test this, we split the Wikipedia list of common misspellings into a train and test set, and calculate the spelling correction

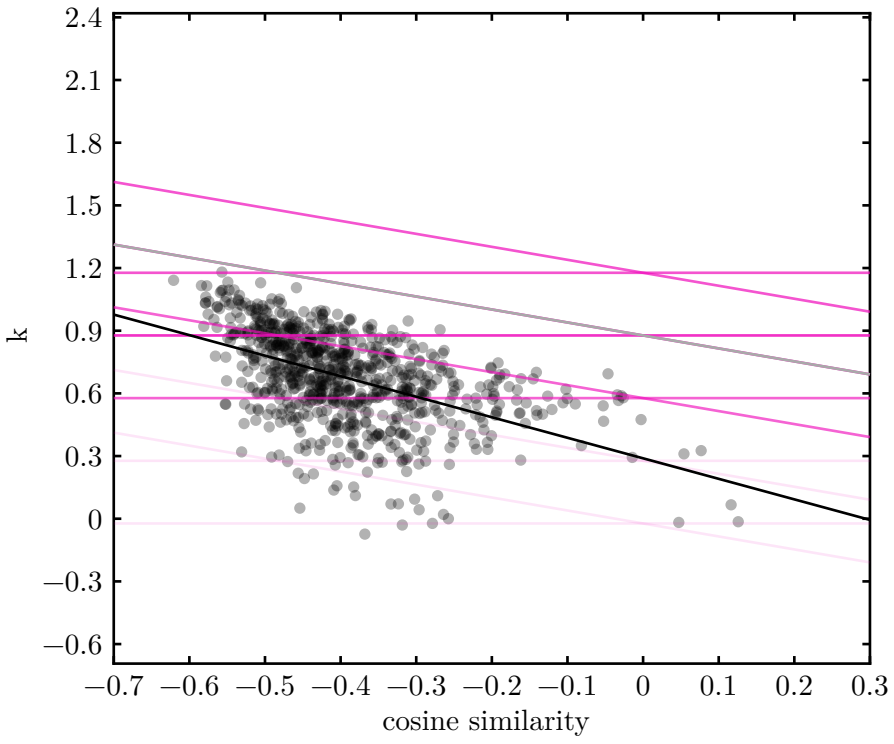

FIGURE 4. Optimal scaling factor k plotted over the cosine similarity between v_{sc} and misspelled words sampled from the Wikipedia list of common misspellings. The linear fit is colored in black, whereas the parameters found through optimization in Rushton's blog post are added in gray. The pink lines show the parameters tested in Table 1.

vector as well as the optimal parameters m and n on the training set. We then evaluat the correction vector with varying parameters on the test set, both for the incorrect words and the correct words. The i2c (incorrect to correct) score shows the ratio of incorrect words corrected to their correct counterpart, wheareas c2c (correct to correct) shows how many correct words were not changed into a different word by the correction. To give a better intuition, the tested parameters are plotted using pink lines, with opacity related to their i2c score.

	m = 0		m = −0.62		m = −2	
n	i2c	c2c	i2c	c2c	i2c	c2c
−0.023	0.004	1	0.004	1	0.426	0.602
0.277	0.009	1	0.009	1	0.694	0.88
0.577	0.554	0.998	0.554	0.998	0.609	0.996
0.877	0.764	0.995	0.764	0.995	0.471	0.996
1.177	0.618	0.995	0.618	0.995	0.339	0.997

TABLE 1. Evaluation of Rushton's approach on the test split of the Wikipedia list for combinations of the parameters m and n. The results do not vary when using a constant function ($m = 0$) instead of the slope determined by Rushton ($m = -0.62$). Because the values for those selections of m do not vary at all, the results for $m = -2$ are included as a check.

As can be seen in Table 1, using the constant function with $m = 0$ is just as effective as when using the slope determined by Rushton. For sanity checking the results, an arbitrary slope of $m = -2$ is included in the results, showing, as expected, a worse effectiveness. This observation falls into line with the run `rushton_const` achieving a higher score than `rushton_re_wiki` in the final evaluation, the only difference being the latter using $m = -0.62$ as used by Rushton, while the former uses $m = 0$. Despite further experiments and visualizations we could not find a definite answer to whether there is an advantage to using a non-constant k depending on the cosine similarity to the spelling correction vector.

4.4 **Rotation instead of Translation**

A detail omitted so far, is the normalization of GloVe vectors before performing the vector calculations described above. Since Rushton always works with normalized GloVe vectors, he also uses normalized vectors when adding the spell correction vector to them. Altough it seems unintuitive, as the relations in the vector space are distorted when all vectors are scaled into a hypersphere by normalizing them, the unnormalized vectors performed worse in some of our experiments.

Thinking about the meaning of going in the direction of the spell correction vector when all starting points lie on a hypersphere, the idea emerged that the spell correction relationship might be representable as a rotation, rather than the translation achived by adding the spell correction vector. Instead of a correction vector,

there could be a rotation matrix transforming the normalized word embeddings of misspellings into the embeddings of correctly spelled words by rotating them around the origin. The problem of finding such a rotation matrix was originally posed in the context of determining the attitude of satellites and became known as *Wahba's problem* [10]. Specifically, the problem is to find an orthogonal matrix R such that the average distance between the embedding w_c of a correct word and the embedding w_i of the matching incorrect spelling rotated by R is minimized.

$$J(R) = \sum_{(c,i) \in \text{ pairs}} \|w_c - Rw_i\|^2$$

The Python library `rowan`[4] provides an implementation of the *Kabsch algorithm* for solving this problem in arbitrary dimensions. Our implementation uses the same procedure as in Rushton's method described above, with the only difference of rotating the vectors of misspelled words instead of adding v_{sc} to them. In the evaluation results shown in Table 2, the rotation approach performs poorly compared to Rushton's approach. However, it does introduce no false corrections for already correct words in this experiment setup, as can be seen by the c2c score of 1, and only a few when evaluated on the Webis QSpell dataset as is shown in Section 5.3.3.

corrector	evaluation set	i2c	c2c
rotate	wikipedia	0.399	1
rotate	rushton	0.189	0.999
rotate	oxford	0.694	1
rushton_const	wikipedia	0.764	0.994
rushton_const	rushton	0.894	0.998
rushton_const	oxford	0.946	1
rushton_re	wikipedia	0.763	0.994
rushton_re	rushton	0.904	0.999
rushton_re	oxford	0.946	1

TABLE 2. The rotation based spell correction approach compared to the translation approaches of Rushton, trained and evaluated on distinct train and test splits of the Wikipedia list of common misspellings

[4]https://rowan.readthedocs.io/en/latest/package-mapping.html

5 Evaluation

The common query datasets like Natural Questions, MS MARCO, and TREC Deep Learning used to test retrieval systems with do not contain annotated spelling mistakes. When trying to measure the effect of spelling mistakes on the retrieval effectiveness, researchers resort to artificially introducing spelling mistakes instead, treating the original queries as the correctly spelled groundtruth. Transformations such as random character insertions, deletions and substitutions, swaps of adjacent letters on the keyboard, and replacing correctly spelled words by common misspellings, are applied to the queries to simulate typos [9].

Because Rushton's approach extracts a list of commonly misspelled words from the GloVe vocabulary, it can only be tested on real world query logs containing exactly these mistakes commonly produced by human writers. The Webis QSpell dataset [1] provides 54,772 manually annotated queries, 9,170 of them having alternative spelling variants. The queries were sampled from the AOL query log released in 2006 and follow the same query length distribution as the whole query log, for queries that are between 3 and 10 words long. Queries can have multiple correct spelling variants, which often also include the original spelling variant as typed by the user. When the Webis QSpell dataset was released in 2017, the performance of the Google Search spelling correction and the Bing Spellcheck API was evaluated on the dataset. To include current state-of-the-art spelling systems for comparison and to see whether Google's and Microsoft's spellchecking improved in the past years, we crawled the whole Webis QSpell dataset again and publish the responses obtained from the search engines alongside this thesis for future research.

5.1 **Google Search**

Apart from providing live suggestions and underlining every misspelled word in red while typing, Google works with four different categories of misspelled queries. At the top of the results page, the phrases "Showing results for", "Including results for", "Did you mean:" and "These are results for" indicate Google's spellchecking found one, or, in rare cases, multiple corrections. In case of multiple corrections, we only stored the first one to make the evaluation easier. Following the evaluation in the Webis QSpell paper, the corrections proposed with the "Did you mean:" label are discarded, as in this case Google is unsure about the correction and is showing the search results for the original query, instead of automatically correcting it. However, treating the "Did you mean:" cases as valid suggestions would result in an even higher effectiveness.

Scraping Google Search is difficult, as Google tries to stop SEO bots from collecting information about their search ranking. However, using a headless Chromium instance controlled using playwright and waiting 30 to 40 seconds between each new search query worked reliably in the end, even from the IP space of a popular rental VPS service in Germany. To get English search results even when accessing Google from an IP address located in Germany, the host language parameter was set to `en`, resulting in the following URL for a given `{query}`: `https://www.google.com/search?hl=en&q={query}&btnG=Google+Search&safe=off&cr=&filter=0&tbas=0`.

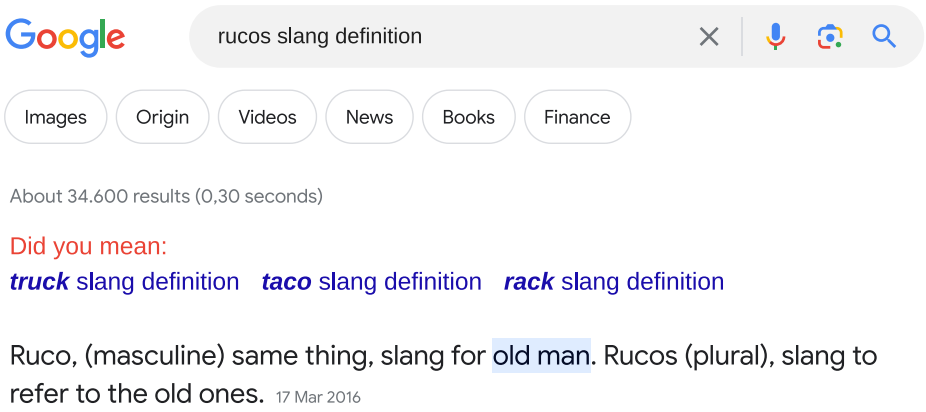

FIGURE 5. Rare occasion of Google providing three spelling corrections.

5.2 Bing Spell Check API

Microsoft's Bing Spell Check API is used as a baseline in many papers [12,13]. It offers two modes of spellchecking: `proof` and `spell`. The former is meant for larger documents and also corrects grammar mistakes, while the latter is meant for returning better search results in web search scenarios. Curiously, Zhuang and Zuccon used the `proof` mode in their evaluation[5], which is why we decided to crawl both of them to find out which one returns the better corrections on the Webis QSpell dataset. In the S1 tier, Microsoft charges $25 per 25,000 transactions, one API call accepting one query to correct at a time. Crawling the Webis QSpell corpus once for each correction mode hence resulted in a total cost of about 100€, which were fortunately covered by the free 200€ Azure credit that is granted when signing up for the first time.

5.3 Results

In the process of experimenting with different parameters for the word embedding based correction approach, evaluations were performed on a 40% train set of the Webis QSpell dataset. An additional 20% split was reserved for validation. The final evaluation was run on the 40% test set to avoid overfitting the hyperparameters to the specific dataset. In correspondence with the Webis QSpell paper, we report Prec@1 as the main measurement of correction effectiveness. It is calculated as the ratio between query corrections that are case-insensitive equal to one of the spelling variants given for each query, divided by the total number of queries. The scenario modeled by this measure is a search engine that has to decide whether to correct a query or leave it untouched, without giving the user a choice for alternatives.

5.3.1 *Baselines*

For comparing the various spell checking approaches described in this thesis, we created multiple baseline runs that either represent best-in-class spell checking algorithms such as the `bing` and `google` runs, or really simple approaches. The `do_nothing` run does not modify the query at all, so the Prec@1 score represents how

[5]https://github.com/ielab/CharacterBERT-DR/blob/main/data/ms_spellchecker.py

many queries in the Webis QSpell dataset contain the original query itself within the set of alternative query variants. Because most of the queries (87.5%) are spelled correctly, this provides a rather strong baseline, as spell checking approaches tend to correct too much. In Table 3, the c2i (correct to incorrect) column indicates how many correct queries were turned into incorrect queries, illustrating this problem. In contrast, the i2c (incorrect to correct) column shows the number of originally incorrect queries that were turned into a query that is part of the alternative spelling variants set of each query.

While `bing_2017` only achieved an effectiveness close to the `do_nothing` baseline, the `bing_2023_spell` run achieves the best Prec@1 score of all tested spell checkers. This indicates that Zhuang and Zuccon chose the seemingly worse baseline to compare against when using the `proof` instead of the `spell` mode, which goes in

model	Prec@1	correct	total	c2i	i2c
bounds_replace_word	0.961	21062	21909	0	1898
bounds_in_glove	0.958	20984	21909	0	1820
bing_2023_spell	0.938	20559	21909	457	1852
google_2017	0.928	20339	21909	172	1347
google_2023_en	0.927	20320	21909	225	1381
bing_2023_proof	0.926	20288	21909	202	1326
rushton_const	0.892	19535	21909	66	437
rushton_re_wiki	0.892	19533	21909	60	429
rushton_orig	0.891	19520	21909	52	408
rushton_re	0.891	19516	21909	66	418
wikipedia	0.879	19255	21909	29	120
bing_2017	0.876	19199	21909	230	265
rotate	0.876	19195	21909	4	35
oxford	0.876	19191	21909	3	30
rushton_linreg_nonorm	0.876	19182	21909	3	21
rushton_linreg_norm	0.875	19171	21909	2	9
do_nothing	0.875	19164	21909	0	0
hunspell	0.687	15057	21909	4498	391
pyspellchecker	0.472	10342	21909	9114	292
birkbeck	0.261	5723	21909	13502	61

TABLE 3. Evaluation on the test split of the Webis QSpell dataset. The c2i score shows the number of originally correct queries that were altered to an incorrect one by the corrector, whereas the i2c score shows the number of originally wrong queries successfully corrected by the spell checker.

line with the documentation that suggests using the `spell` mode for query spelling correction. Google did not improve over the 2017 run, but achieved a similar result in 2023. It should be noted that the Google and Bing spell checkers are the only included approaches that access the whole surrounding query for correcting a word, as all other approaches operate on single words only in a context-free manner.

Since these context-free spell checking approaches can be reduced to lookup tables of (*incorrect, correct*) word pairs, we included multiple existing lists of misspelled words in the evaluation. The `wikipedia` run applies the Wikipedia list of common misspellings, also used for training the spell correction vector. The `oxford` list is the one originally used by Rushton for building the first spell correction vector, and `birkbeck`[6] is another large collection of spelling mistakes from handwritten texts. The `rushton_orig` run uses the correction table Rushton created as a final result of his spell correction, and therefore results in the same score as Rushtons original implementation.

5.3.2 *Traditional Spell Checkers*

Both `pyspellchecker` and `hunspell` result in Prec@1 scores below the `do_nothing` baseline because of the many correct queries they overcorrect into an incorrect one. This is due to web search queries containing numerals, abbreviations, entity names and Internet slang not found in dictionaries and usually not encountered in written text. They are therefore unsuitable for correcting search queries out of the box, but no attempt was made to reduce the strictness of their behavior. It should be noted that Hunspell is meant to be used in a writing scenario where the user selects one of the suggested correction candidates, instead of automatically selecting the first result automatically as done in our evaluation.

5.3.3 *Word Embedding-based Spell Checkers*

The `rushton_re` run is our reimplementation of Rushton's approach and performs similar to the `rushton_orig` run, which uses the substitution table provided by Rushton. By precisely selecting a few common misspellings and finding the correct spelling of those, the spell checking runs based on Rushton's approach overall avoid the problem of correcting too strictly. The `rushton_re_wiki` run uses a spell correction vector trained using the Wikipedia list of common misspellings, showcasing that the list is usable for training a correction vector and that Rushton's approach does not depend on the two-step process he used to create a refined correction vector. The `rushton_linreg` runs show that using the scaling factor k determined by the linear fit does not improve the correction accuracy over using a constant k. The effect of normalizing all the GloVe vectors before calculating the parameters m and n for determining k and adding v_{sc} does not appear to be large when comparing the `rushton_linreg_nonorm` to the `rushton_linreg_norm` run.

To provide an upper bound for how good single word based correction methods could get (when also using the word's context), the oracle run `bounds_replace_word` gets every query right that has the same number of words in the original query as in the correction, or that includes the original query as a valid variant. As the Prec@1 score suggests, only 3.9% of queries contain mistakes introduced by spaces.

[6] https://www.dcs.bbk.ac.uk/~ROGER/corpora.html

All the word embedding based approaches presented are limited by the requirement that both the misspelled word and the correction have to be included in the 2.2 million pre-trained GloVe embeddings. The `bounds_in_glove` run provides an oracle that corrects every query where the correct and incorrect word are contained in the GloVe vocabulary, and assumes the word is correct if it does not have an embedding, showcasing the maximum score a GloVe based corrector could achieve.

The rotation based correction approach falls behind compared to Rushton's method, but it works in principle and might just need a different candidate selection process than the one Rushton developed for his method. Since this thesis explored translation based approaches and a rotation based approach separately, it might be interesting to explore learning a linear map between the incorrect and correct word vectors, which would combine rotation and translation in a single transformation.

6 Conclusions

In this thesis, we recreated the spell correction approach described by Rushton and showed it can be used to correct real-world web search queries. This correction task poses new challenges compared to correcting text documents, which is why established spell correctors for documents like Hunspell are unsuited for it. While the relationship between incorrect and correct spelling variants is undoubtably encoded in the pre-trained GloVe vectors, it remains unclear what is the best approach to extract this information. Although we tried to explore the effect every step in the algorithm has on the correction effectiveness in multiple experiments, we could not get a clear understanding on which procedures actually benefit the effectiveness. Considering we were unable to further improve on the accuracy of the method provided by Rushton, it could be argued Rushton already found the best possible way of using the encoded relationships. However, there are more advanced machine learning methods left to explore, that might be able to capture the relationship between the word vectors in a better way.

Acknowledgements

I would like to thank my supervisors Prof. Matthias Hagen and Ines Zelch for their ongoing support, feedback, and patience. I am grateful for the many times in which Maik Fröbe supported me when facing challenges using the Webis infrastructure, and for the discussion we had when I felt stuck. Lastly, I want to thank the whole Webis group at Jena for the lovely time I had during their weekly seminar, and especially at their Christmas party.

References

[1] Matthias Hagen, Martin Potthast, Marcel Gohsen, Anja Rathgeber, and Benno Stein. 2017. A Large-Scale Query Spelling Correction Corpus. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, 2017. ACM, 1261–1264. https://doi.org/10.1145/3077136.3080749

[2] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, 2020. Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.550

[3] Mark D. Kernighan, Kenneth Ward Church, and William A. Gale. 1990. A Spelling Correction Program Based on a Noisy Channel Model. In *13th International Conference on Computational Linguistics, COLING 1990, University of Helsinki, Finland, August 20-25, 1990*, 1990. 205–210. Retrieved from https://aclanthology.org/C90-2036/

[4] Tomás Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic Regularities in Continuous Space Word Representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, 2013. The Association for Computational Linguistics, 746–751. Retrieved from https://aclanthology.org/N13-1090/

[5] Peter Norvig. 2007. How to Write a Spelling Corrector. Retrieved from https://norvig.com/spell-correct.html

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014. ACL, 1532–1543. https://doi.org/10.3115/V1/D14-1162

[7] Ed Rushton. 2018. A Simple Spell Checker Built from Word Vectors. Retrieved from https://edrushton.medium.com/a-simple-spell-checker-built-from-word-vectors-9f28452b6f26

[8] Victor Shepelev. 2021. Rebuilding the Spellchecker. Retrieved from https://zverok.space/spellchecker.html

[9] Georgios Sidiropoulos and Evangelos Kanoulas. 2022. Analysing the Robustness of Dual Encoders for Dense Retrieval Against Misspellings. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, 2022. ACM, 2132–2136. https://doi.org/10.1145/3477495.3531818

[10] Grace Wahba. 1965. A Least Squares Estimate of Satellite Attitude. *SIAM Review* 7, 3 (1965), 409–410. https://doi.org/10.1137/1007077

[11] Shengyao Zhuang and Guido Zuccon. 2021. Dealing with Typos for BERT-based Passage Retrieval and Ranking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, 2021. Association for Computational Linguistics, 2836–2842. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.225

[12] Shengyao Zhuang and Guido Zuccon. 2022. CharacterBERT and Self-Teaching for Improving the Robustness of Dense Retrievers on Queries with Typos. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, 2022. ACM, 1444–1454. https://doi.org/10.1145/3477495.3531951

[13] Shengyao Zhuang, Linjun Shou, Jian Pei, Ming Gong, Houxing Ren, Guido Zuccon, and Daxin Jiang. 2023. Typos-aware Bottlenecked Pre-Training for Robust Dense Retrieval. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region, SIGIR-AP 2023, Beijing, China, November 26-28, 2023*, 2023. ACM, 212–222. https://doi.org/10.1145/3624918.3625324

Eigenständigkeitserklärung

1. Hiermit versichere ich, dass ich die vorliegende Arbeit – bei einer Gruppenarbeit die von mir zu verantwortenden und entsprechend gekennzeichneten Teile – selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich trage die Verantwortung für die Qualität des Textes sowie die Auswahl aller Inhalte und habe sichergestellt, dass Informationen und Argumente mit geeigneten wissenschaftlichen Quellen belegt bzw. gestützt werden. Die aus fremden oder auch eigenen, älteren Quellen wörtlich oder sinngemäß übernommenen Textstellen, Gedankengänge, Konzepte, Grafiken etc. in meinen Ausführungen habe ich als solche eindeutig gekennzeichnet und mit vollständigen Verweisen auf die jeweilige Quelle versehen. Alle weiteren Inhalte dieser Arbeit ohne entsprechende Verweise stammen im urheberrechtlichen Sinn von mir.

2. Diese Arbeit, sowie der für diese Arbeit geschriebene Programmcode, ist vollständig ohne den Einsatz generativer KI-Anwendungen entstanden.

3. Ich versichere des Weiteren, dass die vorliegende Arbeit bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt wurde oder in deutscher oder einer anderen Sprache als Veröffentlichung erschienen ist.

4. Mir ist bekannt, dass ein Verstoß gegen die vorbenannten Punkte prüfungsrechtliche Konsequenzen haben und insbesondere dazu führen kann, dass meine Prüfungsleistung als Täuschung und damit als mit „nicht bestanden“ bewertet werden kann. Bei mehrfachem oder schwerwiegendem Täuschungsversuch kann ich befristet oder sogar dauerhaft von der Erbringung weiterer Prüfungsleistungen in meinem Studiengang ausgeschlossen werden

Jena, den 27. April 2024