Embedding-based Query Spelling Correction

Ines Zelch[1,2], Gustav Lahmann[1] and Matthias Hagen[1]

[1]Friedrich-Schiller Universität Jena
[2]Universität Leipzig

Abstract

For many retrieval systems, correcting spelling errors in the queries that searchers submit is an essential step of query understanding. Inspired by a blog post on spelling correction from 2018, we implement and analyze a simple embedding-based query spelling correction approach, and we compare the impact on retrieval effectiveness to Hunspell-, pyspellchecker-, and GPT-3.5-based spelling correction. Our experiments with BM25 and PL2 on corpora of the TIRA / TIREx evaluation platform show that especially Hunspell and pyspellchecker are much worse than a do-nothing baseline that does not even try to correct errors. The embedding-based approach comes closest to the baseline but also does not correct many errors, while the GPT-3.5-based approach corrects more errors but at the same time already changes many correct queries and thus worsens the overall retrieval effectiveness. As a software contribution, we provide the four spelling correctors in a Docker image so that they can be easily included as components in retrieval systems and retrieval pipelines.

Keywords

Query spelling correction, Query understanding, Word embeddings, Information retrieval, Web search

1. Introduction

When searchers type a query to express an information need, it can easily happen that spelling errors are included—for instance, by "fat-fingering" on the keyboard or by not knowing how to spell a word. Previous studies showed that about 10–20% of queries contain some spelling errors [1, 2, 3, 4]. Thus, one of the earliest steps in a search engine's query understanding pipeline often is spelling correction.

Typical spelling errors include missing or additional characters (deletion, insertion), missing or additional spaces and special characters, wrong characters (substitution), and swapped characters (transposition); Table 1 shows some examples. In a retrieval pipeline, query spelling errors may lead to several problems ranging from tokenization issues (e.g., for missing or additional spaces), over issues with stemming / lemmatization or entity linking (e.g., misspellings may not match stemming rules or gazetteer entries), to problems with lexically matching a misspelled query to relevant documents.

In this paper, inspired by a 2018 blog post [5], we implement and analyze an embedding-based query spelling correction approach. We compare it to Hunspell-, pyspellchecker-, and GPT-3.5-based spelling correction with respect to the impact on actual retrieval effectiveness. As a software contribution to the WOWS 2024 workshop [6], we additionally provide a Docker image with the four approaches as easy-to-use components for retrieval systems and retrieval pipelines.

2. Related Work

Spelling errors are either so-called real-word errors (the misspelling is another existing word like 'their' instead of 'there') or non-word errors (a misspelling that is not in a dictionary) [7]. Automatic spelling correction approaches usually first detect spelling errors and then rank some suggested correction candidates [8, 9]. While non-word errors can be detected via unsuccessful dictionary lookups, real-word errors are harder to detect as they usually require to consider a word's context [7].

Early approaches for query spelling correction were based on query log analyses [10, 11] but later also included n-gram models [12] and discriminative approaches like SVMs [13]. In the context of

WOWS'24: 1st International Workshop on Open Web Search, March 28, 2024, Glasgow, Scotland
ines.zelch@uni-jena.de (I. Zelch); gustav.lahmann@uni-jena.de (G. Lahmann); matthias.hagen@uni-jena.de (M. Hagen)
0009-0005-2659-5326 (I. Zelch); 0000-0002-9733-2890 (M. Hagen)

Table 1
Types of spelling errors with examples from Wikipedia's Lists_of_common_misspellings.

Type	Misspelling		Correction
Deletion	corosion	→	corrosion
Insertion	occurr	→	occur
Space	abouta	→	about a
Special character	isnt	→	isn't
Substitution	grammer	→	grammar
Transposition	rewriet	→	rewrite

neural retrieval models—that also have problems with misspelled queries—, some recent studies aim at not applying query spelling correction prior to the actual retrieval but instead to increase the models' robustness against misspellings by including simulated typos in the models' training data [14, 15, 16]. Yet, based on these studies' results, query spelling correction still seems to yield better retrieval effectiveness.

As for the accuracy of detecting and correcting query spelling errors, a 2017 study by Hagen et al. [4] on a corpus of about 55,000 queries (9200 with errors) showed that a do-nothing baseline (i.e., not even trying to correct any query) is a rather strong baseline (> 80% of the queries have no spelling errors) that, back in 2017, was only beaten by Google's query spelling correction.

3. Approach

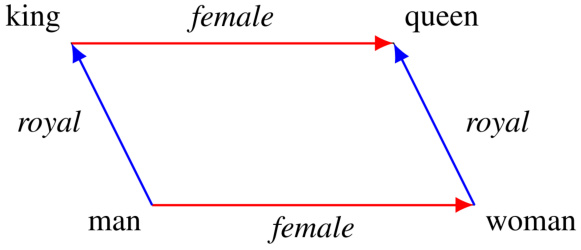
In 2018, Rushton described an idea for general spelling correction based on a "correction vector" in a word embedding space [5]. Word embeddings are dense contextualized vector representations of words and usually are trained on large text corpora. The idea of early embedding approaches like GloVe [17] was that words with a similar meaning probably occur in similar textual contexts so that they should end up in a similar region of the embedding space. Such embeddings were then also promoted as allowing operations like the famous analogy "king is to queen as man is to woman" [17]:

$$\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} \approx \overrightarrow{queen}\,.$$

As illustrated in Figure 1, the vector representation of 'king' lies in a similar direction from 'man' as the representation of 'queen' lies to 'woman' (a "royal" direction / vector), and the representation of 'queen' lies in a similar direction from 'king' as 'woman' lies to 'man' (a "female" vector).

Rushton [5] suggested to use GloVe word embeddings to detect and correct spelling errors. His first idea was that the embedding of an incorrectly spelled word should be close to the embedding of the respective correctly spelled word, so that a correction for a misspelling could be found by looking at the neighbors of the misspelling's embedding. However, instead of the correct spelling being embedded close to a misspelling, Rushton found that the nearest neighbors of a misspelling's embedding actually are embeddings of other misspellings of the same or synonymous words. He then hypothesized that the information about the correctness of a spelling is already implicitly encoded in a word's GloVe embedding during training. As one design goal of the GloVe embedding approach was to encode semantic and syntactic properties of a word as linear substructures in the embedding vector

Figure 1: Illustrated embedding operations (Figure 1 of Ethayarajh et al. [18]).



space, Rushton realized that a misspelling's embedding differs in some dimensions from the respective correct word's embedding, while being almost identical in the others. When calculating the difference vector of a misspelling and the correct word, the dimensions encoding the misspelling property have the highest values. Rushton thus suggested to average the difference vectors for many misspellings to find a common "spelling correction" vector similar to the "royal" or "female" vectors in the above king–queen example and to then basically calculate respective (mis-)spelling "analogies" like:

$$\overrightarrow{computer} - \overrightarrow{compouter} + \overrightarrow{extraordianry} \approx \overrightarrow{extraordinary}\,.$$

Starting with an Oxford Dictionary list of common misspellings and corrections,[1] Rushton calculated a spelling correction vector $\vec{sc}$ by averaging the differences of the respective pairs of GloVe embeddings. As for the actual spelling correction, Rushton tried to add $\vec{sc}$ to the embedding of a potentially misspelled word (in our case, we simply do it for every query word), and to then search for the cosine similarity-wise nearest neighbor as the respective correct word (also using some further heuristics that check for plural forms or common suffixes). In experiments, Rushton found that corrections can be found better when linearly stretching or shortening $\vec{sc}$ based on the angle between a potential misspelling's embedding and $\vec{sc}$. He empirically optimized the linear transformation's parameters and then suggested to use the transformed $\vec{sc}_{opt}$ as the basis of an embedding-based spelling correction approach.

For our re-implementation of Rushton's idea, we use 300-dimensional GloVe embeddings[2] of the 2 million most frequent words trained on 840 billion Common Crawl tokens [17]. We calculate Rushton's exact spell correction vector $\vec{sc}$ from his initial data and we use the linear transformation he found to optimize the spelling correction to arrive at $\vec{sc}_{opt}$. In a further pre-processing to speed up the actual query spell correction, we create an error correction hash table of words and their "best" spell correction by adding $\vec{sc}_{opt}$ to each of the 2 million embeddings and looking for the respective nearest neighbor. To reduce the size of the table, we remove pairs where the nearest-neighbor "correction" actually has the same spelling as the original word, and thus only store pairs with different spelling in the table. To spell-correct a query, we lowercase all words, and check for each individually whether the hash table has a match. Words that are not found in the hash table, are simply considered correct.

We compare the embedding-based approach to Hunspell-, pyspellchecker-, and GPT-3.5-based approaches. Hunspell[3] is an often used spell checker (e.g., OpenOffice and Mozilla), while pyspellchecker[4] is based on a blog post by Peter Norvig;[5] from simple edits of a word (Levenshtein distance), it chooses the one with the highest frequency in a word frequency list. For the GPT-3.5-based approach, we use the prompt: "You are a spelling corrector for search queries. You respond always exactly with the corrected search query. Please correct all spelling errors in the query: <query>".

Our implementations of these query spelling correction approaches are freely accessible on GitHub.[6] Listing 1 shows a code snippet that applies the embedding-based spelling correction (`rushton`) and evaluates it for retrieval with BM25 and PL2 on the ANTIQUE dataset.

4. Evaluation

For our experiments, we run the query spelling correction approaches on the queries of the ANTIQUE dataset [19] and on the TREC 2019 and 2020 Deep Learning tracks [20, 21] before submitting the queries to BM25 and PL2. Our experiments were conducted on the TIRA [22] / TIREx [23] evaluation platform.

Table 2 shows the retrieval effectiveness results (nDCG@10). As kind of an upper bound for using any of the spell correctors, we also give the results of an "oracle" that always chooses the nDCG@10-wise best output of the do-nothing baseline and of the four spell correctors. For all three datasets, the oracle

[1] languages.oup.com/
[2] https://nlp.stanford.edu/projects/glove/
[3] github.com/hunspell/hunspell
[4] pypi.org/project/pyspellchecker/
[5] norvig.com/spell-correct.html
[6] https://github.com/OpenWebSearch/wows24-query-spelling-correction

Listing 1: Code snippet to run and evaluate the embedding-based query spelling correction (`rushton`) for the retrieval models BM25 and PL2 on the ANTIQUE dataset.

```
from tira.third_party_integrations import ensure_pyterrier_is_loaded()
import pyterrier as pt
from tira.rest_api_client import Client
tira = Client()
ensure_pyterrier_is_loaded

data = 'antique-test-20230107-training'
dataset = pt.get_dataset(f'irds:ir-benchmarks/{data}')
index = tira.get_run_output('ir-benchmarks/tira-ir-starter/Index (tira-ir-starter-
    pyterrier)', data) + '/index'
index = pt.IndexFactory.of(index)

bm25 = pt.BatchRetrieve(index, wmodel="BM25")
pl2 = pt.BatchRetrieve(index, wmodel="PL2")

tokeniser = pt.autoclass("org.terrier.indexing.tokenisation.Tokeniser").getTokeniser()
def pt_tokenise(text):
    return ' '.join(tokeniser.getTokens(text))

rushton = tira.pt.transform_queries('workshop-on-open-web-search/qspell/rushton', data)
toks = pt.apply.query(lambda i: pt_tokenise(i['query']))
sys = [bm25, pl2, rushton >> toks >> bm25, rushton >> toks >> pl2]
names = ['BM25', 'PL2', 'rushton >> BM25', 'rushton >> PL2']

pt.Experiment(sys, dataset.get_topics('query'), dataset.get_qrels(), ['ndcg_cut.10'],
    names=names, verbose=True)
```

Table 2
Retrieval effectiveness (nDCG@10) of BM25 and PL2 with the different query spelling correction approaches on the ANTIQUE dataset and on the TREC 2019 and 2020 Deep Learning tracks [19, 20, 21].

	ANTIQUE		TREC DL '19		TREC DL '20	
Spell Correction	BM25	PL2	BM25	PL2	BM25	PL2
Oracle	0.52	0.50	0.48	0.47	0.51	0.50
None	0.51	0.49	0.48	0.47	0.49	0.48
Rushton	0.51	0.49	0.48	0.47	0.49	0.48
GPT-3.5	0.49	0.47	0.48	0.47	0.48	0.47
Hunspell	0.48	0.46	0.36	0.34	0.42	0.41
pyspellchecker	0.43	0.41	0.31	0.30	0.33	0.32

effectiveness is a bit better than not correcting any error (sometimes only in the third decimal, though). The do-nothing baseline on average is better than the individual correction approaches among which the embedding-based approach (line 'Rushton' in the table) is best and substantially better than the "standard" Hunspell- and pyspellchecker-based approaches. The corrections of GPT-3.5 also yield a worse retrieval effectiveness than the embedding-based approach. On TREC DL '20, the embedding-based approach is as effective as the do-nothing baseline (as both do not change any query), while on TREC DL '19, the embedding-based approach is slightly better than doing nothing and as effective as the oracle. Still, on TREC DL '19, the oracle and the embedding-based approach only change one query (`why did the us volunterilay enter ww1`) and, interestingly, not even correct the actual spelling error '`volunterilay`', but change the abbreviation '`ww1`' to the more general term '`war`'.

Possible reasons for the do-nothing baseline overall being so close to the oracle while some correction approaches are much worse are (1) that the queries of the TREC DL tracks hardly contain misspellings,

Table 3
Number of spelling errors per type in the considered datasets (ANTIQUE: 200 queries; TREC DL '19: 43 queries; TREC DL '20: 54 queries); number of corrected queries using Rushton's approach (R), Hunspell (H), pyspellchecker (P), and GPT-3.5 (G); and the total number of changed queries per approach.

	ANTIQUE					TREC DL '19					TREC DL '20				
	#errors	corrected				#errors	corrected				#errors	corrected			
Error Type		R	H	P	G		R	H	P	G		R	H	P	G
Deletion	14	4	6	5	12	-	-	-	-	-	-	-	-	-	-
Insertion	5	1	1	1	4	1	-	1	1	1	-	-	-	-	-
Space	13	1	1	2	11	-	-	-	-	-	-	-	-	-	-
Special character	19	-	2	6	9	1	-	-	-	-	3	-	-	-	-
Substitution	8	1	2	2	8	1	-	1	1	1	1	-	-	-	-
Transposition	4	2	2	1	4	-	-	-	-	-	-	-	-	-	-
Incorrect changes		0	52	75	42		1	12	15	5		0	11	23	15

(2) that the correction approaches also do not correct many misspellings, and (3) that the correction approaches often even "worsen" queries by changing correctly spelled words. Table 3 gives the respective statistics of spelling errors per error type (manually identified), of the number of corrected errors per type and approach, and of the number of incorrectly changed queries per approach. The most frequent error are missing apostrophes (e.g., '`don t`') probably introduced by some query parser before logging.

Among the correction approaches, the GPT-3.5-based one has the most corrections. But it also often incorrectly changes queries which leads to an overall worse average retrieval effectiveness than the embedding-based approach that corrects hardly any of the misspellings but—just like the do-nothing baseline—also does not worsen most of the already correct queries. The many incorrect changes of the Hunspell- and pyspellchecker-based approaches explain their bad overall nDCG impact. For example, on the ANTIQUE dataset with 63 misspellings in 53 of the 200 queries, the embedding-based approach corrects 9 errors and has no incorrect changes, whereas Hunspell and pyspellchecker correct 14 and 17 errors, respectively, but at the same time incorrectly change 52 and 75 queries.

Inspecting the approaches' changes in more detail, we found that GPT-3.5 often appends question marks to queries formulated as questions and corrects the queries' grammar (e.g., why `*does* hot air rise up*?*`). These changes might not be incorrect, but they do not help BM25 or PL2 during retrieval. Some few incorrect changes and artifacts like '`corrected search query`' reduce the retrieval effectiveness for GPT-3.5. At the same time, GPT-3.5 is the only approach that corrects misspellings in person names like '`natalie holloway`' → '`natalee holloway`' or in product names like '`citriscidal`' → '`citricidal`'. Some "normal" typos like '`prsented`' are corrected by all four approaches, while only the embedding- and GPT-3.5-based approaches correct the query '`how can i keep my rabit indoors`'. Hunspell and pyspellchecker change '`rabit`' to '`barit`' or '`habit`' instead. Other incorrect changes by Hunspell and pyspellchecker are unrelated words chosen as "correction" (e.g., '`wifi`' to '`wife`'). For pyspellchecker, for example, this phenomenon can be explained by the use of the word frequency list that leads to replacing unknown or uncommon words with more frequent lexically similar words—with a negative impact on the retrieval results. This does not happen for the embedding-based approach as unknown words (i.e., words without GloVe embeddings) are considered as correctly spelled—with the downside that many typos are not part of the GloVe vocabulary.

Limitations Even though the embedding-based query spelling correction achieves acceptable results in the evaluation, it also comes with some severe limitations. First, the approach can only correct errors for which both the misspelled word and the correct word have a corresponding GloVe embedding. Misspelled words where either the misspelling or the correct word were not present in the GloVe embedding training data cannot be corrected. Second, the embedding-based correction works on single words only and thus cannot detect errors involving missing or additional spaces (either the correction or the misspelling would be at least two words). And third, again due to the single-word characteristic of the embeddings, context-dependent spelling errors (i.e., real-word errors) cannot be corrected.

5. Conclusion and Future Work

In this paper, we analyzed four query spelling correction approaches that we implemented as a software contribution to the WOWS 2024 workshop [6]. Among the approaches, an embedding-based one yields better retrieval results than the Hunspell-, pyspellchecker-, and GPT-3.5-based approaches. Still, on average, all approaches yield worse retrieval results than a simple do-nothing baseline that does not change any query. One reason is that the spell correctors tend to also change already correct queries and thereby worsen the retrieval effectiveness.

As hardly any of the TREC Deep Learning track queries contain misspellings, it could be an interesting idea for a further study to artificially create variants of the queries with some realistic typos and to measure the retrieval impact on these. Another interesting direction to further analyze the strengths and weaknesses of embedding-based query spelling correction is to test other embedding approaches (e.g., word2vec or fastText) and to combine embedding-based correction with other approaches in some system that might try to choose the best correction approach on a per-query basis.

Acknowledgments

Partially supported by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070014 (OpenWebSearch.eu, https://doi.org/10.3030/101070014).

References

[1] H. Dalianis, Evaluating a Spelling Support in a Search Engine, in: Proceedings of NLDB, 2002, pp. 183–190.
[2] S. Cucerzan, E. Brill, Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users, in: Proceedings of EMNLP 2004, 2004, pp. 293–300.
[3] A. Z. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, J. Yuan, Online Expansion of Rare Queries for Sponsored Search, in: Proceedings of WWW, 2009, pp. 511–520.
[4] M. Hagen, M. Potthast, M. Gohsen, A. Rathgeber, B. Stein, A Large-Scale Query Spelling Correction Corpus, in: Proceedings of SIGIR, 2017, pp. 1261–1264.
[5] E. Rushton, A Simple Spell Checker Built from Word Vectors, Blog post on Medium, 2018. URL: https://web.archive.org/web/20231204160726/https://edrushton.medium.com/a-simple-spell-checker-built-from-word-vectors-9f28452b6f26.
[6] S. Farzana, M. Fröbe, M. Granitzer, G. Hendriksen, D. Hiemstra, M. Potthast, S. Zerhoudi, 1st International Workshop on Open Web Search (WOWS), in: Proceedings of ECIR, 2024.
[7] D. Hládek, J. Staš, M. Pleva, Survey of Automatic Spelling Correction, Electronics 9 (2020) 1670.
[8] K. Kukich, Techniques for Automatically Correcting Words in Text, ACM Comput. Surv. 24 (1992) 377–439.
[9] F. A. Pirinen, K. Lindén, State-of-the-Art in Weighted Finite-State Spell-Checking, in: Proceedings of CICLing, 2014, pp. 519–532.
[10] F. Ahmad, G. Kondrak, Learning a Spelling Error Model from Search Query Logs, in: Proceedings of HLT/EMNLP, 2005, pp. 955–962.
[11] M. Li, M. Zhu, Y. Zhang, M. Zhou, Exploring Distributional Similarity Based Models for Query Spelling Correction, in: Proceedings of ACL, 2006.
[12] F. Ahmed, E. W. D. Luca, A. Nürnberger, Revised N-Gram based Automatic Spelling Correction Tool to Improve Retrieval Effectiveness, Polibits 40 (2009) 39–48.
[13] H. Duan, Y. Li, C. Zhai, D. Roth, A Discriminative Model for Query Spelling Correction with Latent Structural SVM, in: Proceedings of EMNLP, 2012, pp. 1511–1521.
[14] G. Sidiropoulos, E. Kanoulas, Analysing the Robustness of Dual Encoders for Dense Retrieval Against Misspellings, in: Proceedings of SIGIR, 2022, pp. 2132–2136.

[15] S. Zhuang, G. Zuccon, Dealing with Typos for BERT-based Passage Retrieval and Ranking, in: Proceedings of EMNLP, 2021, pp. 2836–2842.
[16] S. Zhuang, G. Zuccon, CharacterBERT and Self-Teaching for Improving the Robustness of Dense Retrievers on Queries with Typos, in: Proceedings of SIGIR, 2022, pp. 1444–1454.
[17] J. Pennington, R. Socher, C. D. Manning, Glove: Global Vectors for Word Representation, in: Proceedings of EMNLP, 2014, pp. 1532–1543.
[18] K. Ethayarajh, D. Duvenaud, G. Hirst, Towards Understanding Linear Word Analogies, in: Proceedings of ACL, 2019, pp. 3253–3262.
[19] H. Hashemi, M. Aliannejadi, H. Zamani, W. B. Croft, ANTIQUE: A Non-factoid Question Answering Benchmark, in: Proceedings of ECIR, 2020, pp. 166–173.
[20] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, E. M. Voorhees, Overview of the TREC 2019 Deep Learning Track, in: Proceedings of TREC, volume abs/2003.07820, 2019.
[21] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, Overview of the TREC 2020 Deep Learning Track, in: Proceedings of TREC, volume 1266, 2020.
[22] M. Fröbe, M. Wiegmann, N. Kolyada, B. Grahm, T. Elstner, F. Loebe, M. Hagen, B. Stein, M. Potthast, Continuous Integration for Reproducible Shared Tasks with TIRA.io, in: Proceedings of ECIR, Berlin Heidelberg New York, 2023, pp. 236–241.
[23] M. Fröbe, J. Reimer, S. MacAvaney, N. Deckers, S. Reich, J. Bevendorff, B. Stein, M. Hagen, M. Potthast, The Information Retrieval Experiment Platform, in: Proceedings of SIGIR, 2023, pp. 2826–2836.