Automatic Spelling Correction as an Information Retrieval Task Using Wikipedia Search Statistics

First Author[1] and Second Author[2]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com

Abstract. Spelling correction utilities have become commonplace during the writing process, however, many spelling correction utilities suffer due to the size and quality of dictionaries available to aid correction. Many terms, acronyms, and morphological variations of terms are often missing, leaving potential spelling errors unidentified and potentially uncorrected. This research describes the implementation of WikiSpell, a dynamic spelling correction tool that relies on the Wikipedia search functionality as the sole source of knowledge to aid misspelled term identification and automatic replacement. Instead of a traditional matching process to select candidate replacement terms, the replacement process is treated as an information retrieval process using harnessing wildcard string matching and search result statistics.

The aims of this research include: 1) the implementation of a spelling correction algorithm that utilizes the wildcard operators in the Wikipedia search API 2) a review of the current spell correction tools and approaches being utilized, and 3) testing and validation of the developed algorithm against the benchmark spelling correction tool, Hunspell. The key contribution of this research is a robust, information retrieval-based spelling correction algorithm that does not require prior training. Results of this research show that the proposed spelling correction algorithm, WikiSpell, achieved comparable results to an industry-standard spelling correction algorithm, Hunspell.

Keywords: Information Retrieval, Wikipedia, Spelling Correction.

1 Introduction

Spelling correction algorithms have become commonplace in many day-to-day applications. Although robust, the downfall of many spell correction algorithms as the training set that is required or the annotated data sets for many algorithms to function [1]. To generate these, manual intervention and labelling is often required by domain experts [2]. The common spelling correction process can be broken into three defined steps. The first of these is the error detection process, where given an input string is validated and individual misspelled words are identified. The second step is to generate a candidate list of terms that can be a suitable replacement for the identified error. The

final step in the process is the ranking of individual candidate terms that can be used for automatic replacement, where the no user intervention is required, and manual replacement, where a user is presented with a collection of replacement terms to choose from.

Brill [3] described spelling error as belonging to one of two categories: typing errors and cognitive errors. Gong et al. [4] defined four common spelling errors as character insertion, permutation, replacement, and removal. As a solution to limited dictionaries and corpora, Wikipedia is an ever-evolving resource, modifications to articles and the inclusion of new articles and the updating of existing articles is daily occurrence [5]. Previously, Wikipedia has be harnessed for a number of different information retrieval tasks such as search query enhancement [6], cross-language information retrieval [7], and question answering [8].

This research describes a novel approach to the automatic identification of misspelled terms by identifying co-occurrences in samples retrieved from the Wikipedia search functionality facilitated by the WikiMedia platform API.[1] Candidate replacement terms are identified through an information retrieval process based on string patterns that utilise available wildcard matching. [2]

In this paper, Section 2 outlines the related work, describing the variation that exists in spelling errors and the limitations to approaches that are currently utilised. Section 3 describes the overall methodology followed for this research. Section 4 describes the implementation of the proposed algorithm, outlining tuning parameters available and subsets of data that are available for manipulation during retrieval. As the process of testing spelling correction varies between implementations, Section 5 outlines the experimentation setup and procedures followed during testing. Section 6 outlines and analyses the results of the testing process. Finally, Section 7 concludes this research and outlines the key findings after the implementation of an information retrieval-based spelling correction algorithm.

2 Related Work

Spelling errors can exist in many forms. Examples of this include terms that have obvious misspellings [9], terms that are correct although in the wrong context [10], and phonetic spelling errors that are typically performed by children [11]. Spelling correction algorithms are highly dependant on having a quality source of language specific corpora to utilize to function correctly, which for less widely used languages becomes an issue as they often do not exist in a complete form [12]. As a solution to this, spelling correction algorithms have been created that utilize dynamic corpora such as Wikipedia for many languages as the list of correct terms is never complete and always growing

[1] https://www.mediawiki.org/wiki/API:Search
[2] https://en.wikipedia.org/wiki/Help:Searching

[13]. In addition to this, grammatical error and correction corpora have also been generated for spelling correction tasks [14].

There is no current standard approach to spelling identification and correction, leaving modern approaches to adopt a variety of implementations including language models, term occurrence statistics, and machine learning. Pirinen et al. [15] described an approach based on weighted finite-states that utilised Wikipedia as a source of English terms. Their approach showed to be faster than the Hunspell[3] algorithm that is widely used in tools such as LibreOffice[4] and Google Chrome.[5] Park et al. [16] described Korean spelling correction using the sequence-to-sequence model. They outline the problem of spelling issues as similar to that of translating from one language into another.

Li et al. [17] described an algorithm that utilizes a transformer-encoder that encodes spelling information and global context information in a neural network. Their solution outperforms the previous state-of-art result by 12.8%. Their approach is designed to not introduce new tokens the original structure is retained for future processing. Haldek et al. [18] described the issue with deep neural networks is that they require annotated data sets that can be expensive to create.

For the proposed algorithm, Wikipedia is used as the sole source of data. Wikipedia provides a wealth of information in both the utilization of content and available statistics. Examples of these content sources include harnessing pageviews for insights [19], page content for the estimation of incidents [20], and utilizing revision history to aid grammatical error correction [21]. Based on these statistics, metrics can easily be derived including semantic relatedness measures [22], ranking and quality assessment [23] and semantic similarity [24]. Although Wikipedia has been used as a source of a priori, but not as a source of statistics to aid spelling identification and correction as described in this paper.

There is currently not a standard approach for the testing of spelling correction algorithms as each utilize different test sets and result ranking approaches [1] [18]. Many papers utilise synthetic spelling errors to aid the testing process [3]. To test the proposed algorithm, a benchmark was required. A gold standard of spelling correction in the English language is Hunspell which was used during this research.

[3] http://hunspell.github.io/
[4] https://www.libreoffice.org/discover/libreoffice/
[5] https://www.google.com/intl/en_ie/chrome/

3 Methodology

To test both Hunspell and the proposed algorithm, a synthetic test query set was designed to implement common spelling errors. These errors included the addition of one or two individual random characters for each term at any position in the string. These errors were introduced into the TREC 2009 Million Query Track 20001-60000 data set[6] that contains 2,000 queries of varying length.

This approach provided both an original query as a reference point and a modified error version of the query for testing. From this set, queries 20001 – 22001 were utilised. These queries contain common user queries to a search engine that include abbreviations, acronyms, and common phrases on a wide variety of topics. Table 1 describes the number of occurrences for each query length in the set.

Query length	Number of occurrences
1	372
2	762
3	552
4	211
5	80
6	14
>7	10

Table 1: Length of original queries.

For these original queries, 1,050 one letter spelling errors and 1,911 two letter spelling errors were introduced. Each error consisted of one random letter between the letters A and Z. 4,951 individual tokens were under analysis with an average of 1.67 spelling errors per query. Additional inserted characters represent the estimation of spelling that users often make or accidently insert when typing. Two was chosen as the max number of characters to insert as >2, the context of the original token can easily be lost. As the proposed algorithm, is experimental, time was not considered as a factor of success due to the time associated to the information retrieval process.

3.1 Error Correction Test Statistics

For each query, precision, recall and accuracy were calculated as shown below:

- **Precision** is defined as: TP / (TP + FP)
- **Recall** is defined as: TP / (TP + FN)
- **Accuracy** is defined as: (TP + TN) / (TP + FP + TN + FN)

TP is defined as the number of works with spelling errors where the algorithm has given the correct suggestion. **FP** is defined as the number of words with or without

[6] https://trec.nist.gov/data/million.query09.html

spelling errors for which the algorithm made suggestions and they result is not needed or is incorrect. **FN** is defined as the number of words with spelling errors that the algorithm did not flag as having errors or did not give suggestions. **TN** is defined as the number of correct words that the algorithm did not flag as having errors and no suggestions were made.

4 Implementation

This section describes the design and implementation of the proposed algorithm, WikiSpell that utilizes the Wikipedia Search API [7] statistics and text results.

4.1 Source of candidate terms

The Wikipedia Search API results are used as source of terms related to a single term in the original sequence of terms under analysis. For any given search result the article *title* and *snippet* were used as sources of terms for a given article. The summary of text is typically 25 terms. Number of results to return was set using *srlimit=100* parameter to limit to a max of 100 results. To avoid overfitting of terms, for any sequence of terms they are tokenized and submitted separately broadening the results collection.

4.2 Weighting Metrics

In this algorithm, two core metrics for relevance are used. The first these is the search result statistics. Given a string *Q*, the number of records returned relates to the number of Wikipedia articles where the string *Q* was found. The total hits were returned using *srinfo=totalhits* API parameter in the Search API. This statistic was used to:

1. Identify that a term has been used before in the English Wikipedia corpus. This also serves to show if the spelling is correct as the number will be higher than those incorrectly spelled in the English language corpus.
2. Identify if a sequence of terms has together been seen in context where the length of a sequence is > 1. The more frequent the use, the higher the chance the sequence is correct.

The hypothesis of this metric is that terms shown in context together provide an indicator of being semantically correct and a viable replacement for incorrect term sequences. Terms that are not frequently shown together would indicate that terms are semantically not a good fit. In addition to this, the Levenshtein distance was also used for small sets of terms to identify if a given term is a good replacement for a candidate term. For a collection of terms {T}, the lower the score for a single term T_n indicates that the term is a closer match.

[7] https://en.wikipedia.org/w/api.php

4.3 Algorithm Implementation

The process of identifying the incorrect spelling of terms and replacing each with the correct term is performed as a complete process in the proposed algorithm, not as individual steps. The algorithm is broken into two distinct steps: 1) the corpus generation process and 2) the candidate generation and selection process. The core algorithm shown in Algorithm 1 is responsible for triggering this process. The algorithm takes a sequence of terms to process shown as *S* that contains terms t_0 to t_n separated by spaces. An arbitrary limit of ten terms per spelling correction run is applied to allow the algorithm to run in a timely manner.

For each of the individual terms *t* inside the sequence **S,** a call is first made to the *generateCorpus()* function which takes in the single term *t* as a parameter. This process is responsible for generating a collection of wildcard variations of the original term *t* with the intent of broadening the morphological variations of the term. This is done by passing each variation of the term to the Wikipedia Search API. The top 100 results returned are collected and stored as a local corpus that can be searched for selecting candidate terms.

Once the corpus generation process has completed, a call is then made to the *generateCandidateReplacement()* process which takes a single parameter titled *t*. This function utilizes and array of functions to select the best candidate to replace a misspelled term with based on the local corpus collected in the previous step.

Algorithm 1: Core process

```
S = Original terms (t0 .. tN)
for each t ∈ S do
   generateCorpus(t)
   generateCandidateReplacements(t)
```

The first function in the algorithm is the *generateCorpus()* function as shown in Algorithm 2, which is passed a single base term *t*. The algorithm is responsible for generating four different permutations of the base term. For each different character position in the term *t* between 0 and the length of *t* the following operations are performed:

i) INSERT - A search wildcard is added which is represented by a single asterisk.
ii) INSERT - Two asterisk characters are inserted.
iii) REPLACE - A single character inside the string is replaced by a single asterisk.
iv) REMOVE - A single character is removed from the original string.

An example of these wildcard string generations for the term *"John"* can be seen in Table 2. All generated variations are prefaced with a single tilde, to encourage the Wikipedia Search API to output search results instead of redirecting to a single article related to the search string.

One asterisk inserted.	One character removed.
~*john	~ohn
~j*ohn	~jhn
~joh*n	~jon
~john*	~joh
Two asterisks inserted.	One character replaced.
~**john	~*ohn
~j**ohn	~j*hn
~joh**n	~joh*
~john**	

Table 2. Example Wildcard corpus generation output for input "*john*".

In the function, a global array titled *generations* is created that is responsible for holding each of the wild card sequences that have been generated from the original term *t*. A variable titled *currentPos* is added that is responsible for holding the current position in the term *t* that is currently being processed. The variable titled *wildcard* holds a reference to the set containing the available wildcards to be entered.

For each individual position in the string *t* from *0* to *n*, where *n* represents the length of the string, the wildcard insertion process is performed. This is started by selecting each wildcard from the set shown as *wc*. A variable titled *newgen* is then created which contains the original string from position 0 up to the current position is appended, followed by the character *wc*, followed by the current position in the string + 1, to the end of the string shown as |t|.

This process is then repeated, for the next string titled *newgen*. The main difference is the omission of the +1, leading to an overwriting of an existing string position value in string *t*. Finally, from position *0* to *curentPos+1* is appended to the *newgen* string followed by currentPos+|t|, removing a single character from the original string in the process. A single character can only be removed as more than one can easily loose the context of the original terms.

Algorithm 2: Corpus generation process

```
Input: t - Individual term from sequence S
function generateCorpus(t)
  global generations = list()
  currentPos = 0
  wildcard = {*, ** }

  for each position in t do
      // Generate wildcards from set incrementally
      for each wc ∈ wildcards do
        // Insert
        newgen = t[0..currentPos] + wc + [currentPos+1..|t| ]
        generations.append("~" + newgen)
```

```
        // Replace
        newgen = t[0..currentPos] + wc + [currentPos..|t| ]
        generations.append("~" + newgen)

        // Removing single characters incrementally
        newgen = t[0..curentPos+1] + t[currentPos+2…|t| ]
        generations.append("~" + newgen)

    currentPos++
```

After the generation process of the sequences has been completed, the process of generating and selecting possible candidate replacement terms can be performed. Algorithm 3 outlines the *generateCandidateReplacements()* function that takes each single term *t* from the original term sequence *S* and held for later comparisons. Then, for each of the individual wildcard sequences showed as *seq* stored inside the global *generations* array from the previous function a call is made to the Wikipedia Search API and the sequence is passed as the query string. The top 100 search results returned are stored as {*R}*. In the search results both the title of the article and the summary of the abstract are extracted. The terms are then tokenized and added to the local array {*T}*.

Due to wildcards being used during the search process, the collection of results will have a marginal relevance to the original search term under consideration. To apply a relevance filter to the terms, a second loop is used to iterate over the terms stored in *T*. For each individual term, the Levensthtein distance is applied between the original query term *t* and the current term t_n. Each of the terms and associated distance scores are appended to the *simvals* array. The distances are then stored in descending order.

During analysis the top five terms proved to be the most relevant to the initial term *t*. For this reason, a filter is applied to retain the top 5 terms from the generated collection of terms and distance scores. For this research n=5. For each of the retained terms, a query is made to the Wikipedia Search API and the current term under analysis is passed. The total number search results are then returned. This is used as a metric to identify if the term is commonly used or not. If the term count is above 500, e.g., indicating that it has been used many times, the term is appended to the *finalVals* array for future processing. The results for a given term *t* are appended to the *resultsPerTerm* array at position *t*.

Once all the terms in the original string have been processed, a second process is started to generate a collection of term sequences using the generated terms and identify the likelihood of the sequence being grammatically correct. To do this, two different variations are used, one which is for terms of length 1 and a second that is utilized for terms greater than 1. These two approaches were created as short queries of 1 term do not need the Cartesian product to be generated. For queries of length 1, the *resultsPerTerm* array is accessed and the individual term *t* is passed as a reference. This returns a collection of terms and associated weights that are appended to the *sequences* array.

For queries with a length greater than 1, the Cartesian product of all results for terms *t0…tn* are calculated. All generated cartesian products are added to the sequences array. The sequence is then passed to the Wikipedia search API and the number of results is returned and stored. A weight by score is then applied to the applied to the results which are stored in the *res* array. To select the final terms to utilize, again two different approaches are utilized. One for short queries of length 1, which is based on the Levenshtein distance between the current sequence and the original term stored in *S*. For longer queries, the top weighted sequence is returned.

Algorithm 3: Generate and select replacement

```
    Input: t (Individual term from S)
    function generateCandidateReplacements (t)
      resultPerTerm = list()
      for each seq ∈ generations do
        make connection to Wikipedia Search API passing seq
        {R} = Select 100 search results
        {T} = terms from {R}

        simvals = arr()
        for each {T} related to t as tn
          dis = levenshtein(t, tn)
          simvals.append(n, dis)
        sortDesc (simvals)
        Select top n terms by weight

        finalVals = arr()
        for term in simvals do
          count = total number of search results for t
          if term > 500:
            finalVals.append(term, count)
        sortByWeightDesc(finalVals)
        resultPerTerm[t] = finalVals

    // Generate sequences of possible replacements and score
    sequences = array()
    if length(S) == 1
      for terms in resultPerTerm[t] do
        score =  number of search results for t
        sequences.append(terms, score)
    else:
      res = cartsianp(resultPerTerm[A x B x …. N])
      for seq in res:
        score =  number of search results for seq
        sequences.append(res, score)

// Final generation of replacements
vals = sortDesc(sequences)
if length(S) == 1
   res = levenshtein(S, vals)
   return lowest(res)
else
    return vals[0] // Return first result e.g. top weighted
```

The selected top terms are utilized as the replacement for incorrect terms in the original query *Q*.

5 Experimentation

The default *en_US* dictionary[8] for Hunspell was utilised for testing, as used in previous studies [11]. The first replacement term from the suggested list of terms for an identified misspelling was utilized. For the proposed algorithm the number of records during consideration was set to 5, the number of search results records included (max) = 100 and the threshold for relevance = 500.

6 Results and Analysis

Table 3 outlines the overall results for 2,000 queries corrected by both the Hunspell and the proposed algorithm providing an even baseline for comparison. As the queries contained a variable number of terms, on the left of the table, the number of terms added for each query size can be seen. The most difficult terms to correct are those that do not contain any context, e.g., single tokens. The proposed algorithm achieved an average precision of 0.533 compared to 0.386 achieved by Hunspell. Between two and five terms long, the results of both Hunspell and the proposed algorithm are comparable. Between six and greater terms, a visible drop can be seen in the precision for both algorithms. The proposed algorithm failed for six and above terms, which can be expected as overfitting to the number of available topics during the information retrieval process could be seen. Although the precision and accuracy greatly improved for the number of terms added (where n=6), this can be attributed to the smaller number of queries being run (14).

	Hunspell	**WikiSpell**	**Hunspell**	**WikiSpell**
# Terms	Precision	Precision	Accuracy	Accuracy
1	0.386	0.533	0.386	0.533
2	0.567	0.563	0.674	0.634
3	0.568	0.591	0.710	0.695
4	0.506	0.571	0.689	0.670
5	0.567	0.564	0.736	0.702
6	0.596	0.0	0.761	0.0
>=7	0.448	0.0	0.687	0.0

Table 3: Precision and recall for each different query length.

[8] https://github.com/elastic/hunspell/tree/master/dicts

In Table 4, the average results for accuracy, precision and recall can be seen for the entire collection of 2,000 queries. Both Hunspell and the proposed algorithm have comparable results.

	Accuracy	**Precision**	**Recall**
Hunspell	0.636	0.527	0.654
WikiSpell	0.631	0.559	0.656

Table 4: Overall average accuracy, precision and recall from 2,000 results.

6.1 Discussion

During the testing process of the benchmark algorithm Hunspell, when the max number of synthetic errors were introduced, e.g., 2 additional characters the algorithm had an issue identifying the correct replacement. Given the term *toilet* represented as *toiletzl*, the suggested replacement was *toilette.* The brand name *volvo* was represented as *volvno* and replaced as *volcano*. The process of appending synthetic errors for the company name *yahoo* which was shown as *yahoogc* was correctly replaced as *yahoo*. Incorrect replacements were often seen when no error was present such as the term *michworks* being replaced with *patchworks*. A lack of terms in the corpus of Hunspell can be easily seen for brand names such as *US airways* which was represented as *usairwaymi* and corrected as *stairway*.

For the website, titled *digg*, which no error was introduced, it was corrected as *dig*, outlining that it did not have knowledge about current brand or service names. Larger queries such as *blount county sheriff department* it was corrected as *bluepoint county sheriff department* which is quite a close match. The larger the number of additional characters added to create an error, the more difficulty the algorithm had. When spaces where missing, the algorithm functioned well to replace it which was be seen with the query *flashyour* which was replaced with *flash your*.

The insertion process was also successful for the name *club pengi* which was replaced with *club penguin*. Other cases of estimation can be seen with the query *Keflex for animals* which was replaced with *reflex for animals*. The algorithm did not seem to notice that the spelling, *Keflex*, was in fact correct. The same can be seen *for supermeds* which no error was introduced but replaced with *supermen*.

For the proposed algorithm WikiSpell, *obama family tree* as *obama frvamily trdee* corrected as *obama family tree.* This outlined that even when two thirds of a query were incorrect, the algorithm still performed well. The single term *toilet* described earlier represented as *toiletzl* was corrected as *toilet* successfully. The name *Euclid* was modified as *euclidxf* and corrected as *Euclid.* As estimation can be seen with the original query *gmat prep classes* that was modified as *gmfat alprep classes* and finally corrected as *fat are classes*. This shows the danger that a single character in an acronym can easily change the entire meaning of an original meaning.

Two characters appended to the acronym *djs* was represented as *djsjv* and corrected as *djs* showing that terms appended to the end of a query pose less of an issue for the algorithm. The four term query *orange county convention center* was modified as *dorange county convention centjrer* and correctly fixed as *orange county convention center*. The three terms *video game artist* was modified as *vidnoeo gamesv azvrtist* and finally completely corrected as *video game artists*. With very little terms close to the name *kodak* when it was represented as *kopdak* it was successfully replaced as *kodak*.

During the retrieval process of Wikipedia data for the proposed algorithm, individual terms were used as the basis for generation even when more than one term was in the original query. The proved to be the most acceptable approach as more than one token routinely returned small collections of results causing the algorithm to prematurely fail. Utilisation of search statistics and content have shown to be a robust solution to replace hand-crafted dictionary corpora as often content and statistics were available. A strong advantage of the proposed algorithm was the harnessing of context available from the source data for sequence validation which many current spelling correction algorithms lack.

Due to the vast quantity of available English data, rare terms were often seen in valid sequences of terms that would not be correct if typical grammatical rules were applied. A core advantage is the availability to correct terms that have recently been added to Wikipedia and the English language which traditionally would be missed from existing corpora.

7 Conclusion

This research proposed the WikiSpell algorithm for the automatic detection and correction of spelling errors in a sequence of terms. Results from this research have shown that the Wikipedia Search API has shown to be effective as a source of candidate terms for spelling correction due to the variety of topics covered in the English Wikipedia. The utilization of Wikipedia English corpus search statistics such as the number of search results as a source for identifying term sequences has shown to be useful for spelling error identification and replacement.

The algorithm proposed in this research, WikiSpell, has shown comparable results to the Hunspell algorithm when used for automatic spelling identification and correction. When working with short sequences of terms, an IR based approach to spelling correction was highly successful due to the dependency on context.

7.1 Future work

To further aid the proposed algorithm, additional focus on the term selection process where a candidate is selected from a small group of terms (n=5) could be enhanced by replacing this with a more dynamic value depending on the collection size and the number of terms under consideration before final selection.

Bibliography

[1] Hládek, D., Staš, J. and Pleva, M.: Survey of Automatic Spelling Correction. Electronics. 9(10): 1670 (2020). https://doi.org/10.3390/electronics9101670

[2] Hagen, M., Potthast, M., Gohsen, M., Rathgeber, A., & Stein, B.: A large-scale query spelling correction corpus. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1261–1264 (2017)

[3] Brill, E. and Moore, R.C.: An improved error model for noisy channel spelling correction. In Proceedings of the 38th annual meeting of the association for computational linguistics, pp. 286-293 (2000)

[4] Gong, H., Li, Y., Bhat, S. and Viswanath, P: Context-sensitive malicious spelling error correction. In: The World Wide Web Conference Proceedings, pp. 2771-2777 (2019)

[5] Lagunes-García, G., Rodríguez-González, A., Prieto-Santamaría, L., del Valle, E.P.G., Zanin, M. and Menasalvas-Ruiz, E : How Wikipedia disease information evolve over time? An analysis of disease-based articles changes. Information Processing & Management. 57(3): 102225 (2020)

[6] Goslin, K. and Hofmann, M.:A Wikipedia powered state-based approach to automatic search query enhancement. Information Processing & Management. 54(4):726-739 (2018)

[7] Cheon, J. and Ko, Y.: Parallel sentence extraction to improve cross-language information retrieval from Wikipedia. Journal of Information Science, 47(2): 281-293. (2021)

[8] Chen, D., Fisch, A., Weston, J. and Bordes, A.: Reading wikipedia to answer open-domain questions. Computing Research Repository (2017)

[9] Gupta, J., Qin, Z., Bendersky, M. and Metzler, D.: Personalized online spell correction for personal search. In The World Wide Web Conference, pp. 2785-2791 (2019)

[10] Mays, E., Damerau, F.J. and Mercer, R.L.: Context based spelling correction. Information Processing & Management, 27(5):517-522 (1999)

[11] O'Neill, E., Young, R., Thiaville, E., MacCarthy, M., Carson-Berndsen, J. and Ventresque, A.:S-Capade: Spelling Correction Aimed at Particularly Deviant Errors. In International Conference on Statistical Language and Speech Processing. pp. 85-96 (2020)

[12] Etoori, P., Chinnakotla, M. and Mamidi, R.: Automatic spelling correction for resource-scarce languages using deep learning. In Proceedings of ACL 2018. Student Research Workshop, pp. 146-152 (2018)

[13] Beeksma, M., Van Gompel, M., Kunneman, F., Onrust, L., Regnerus, B., Vinke, D., Brito, E., Bauckhage, C. and Sifa, R.: Detecting and correcting spelling errors in high-quality Dutch Wikipedia text. Computational Linguistics in the Netherlands Journal. 8:122-137. (2018)

[14] Grundkiewicz, R. and Junczys-Dowmunt, M.: The wiked error corpus: A corpus of corrective wikipedia edits and its application to grammatical error correction. In International Conference on Natural Language Processing, pp. 478-490 (2014)

[15] Pirinen, T.A. and Lindén, K.: State-of-the-art in weighted finite-state spell-checking. In Proceddings of the International Conference on Intelligent Text Processing and Computational Linguistics, Berlin, Heidelberg. (2014)

[16] Park, C., Kim, K., Yang, Y., Kang, M. and Lim, H.: Neural spelling correction: translating incorrect sentences to correct sentences for multimedia. Multimedia Tools and Applications, pp.1-18. (2020)

[17] Li, X., Liu, H. and Huang, L., Context-aware Stand-alone Neural Spelling Correction. In arXiv preprint arXiv. (2020)

[18] Hládek, D., Staš, J. and Pleva, M.: Survey of Automatic Spelling Correction. Electronics. 9(10), 1670 (2020). https://doi.org/10.3390/electronics9101670

[19] Vardi, E., Muchnik, L., Conway, A. and Breakstone, M.: WikiShark: An Online Tool for Analyzing Wikipedia Traffic and Trends. In Companion Proceedings of the Web Conference. (2021)

[20] De Toni, G., Consonni, C. and Montresor, A.: A general method for estimating the prevalence of influenza-like-symptoms with Wikipedia data. PloS. 16(8) (2021)

[21] Boyd, A.: Using Wikipedia edits in low resource grammatical error correction. In Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text. (2018)

[22] Karve, S., Shende, V. and Hople, S.: Semantic Relatedness Measurement from Wikipedia and WordNet Using Modified Normalized Google Distance. In *Data Analytics and Learning*. *(*2019)

[23] Lewoniewski, W., Węcel, K. and Abramowicz, W.: Multilingual ranking of Wikipedia articles with quality and popularity assessment in different topics. Computers. 8(3):60 (2019)

[24] Hussain, M.J., Wasti, S.H., Huang, G., Wei, L., Jiang, Y. and Tang, Y.:A n approach for measuring semantic similarity between Wikipedia concepts using multiple inheritances. Information Processing & Management. 57(3): 102188 (2020)