Between Sound and Spelling: Combining Phonetics and Clustering Algorithms to Improve Target Word Recovery

Marcos Zampieri[1], Renato Cordeiro de Amorim[2]

[1]Saarland University, Germany
[2]Birkbeck University of London and Glyndŵr University, United Kingdom
marcos.zampieri@uni-saarland.de,r.amorim@glyndwr.ac.uk

Abstract. In this paper we revisit the task of spell checking focusing on target word recovery. We propose a new approach that relies on phonetic information to improve the accuracy of clustering algorithms in identifying misspellings and generating accurate suggestions. The use of phonetic information is not new to the task of spell checking and it was used successfully in previous approaches. The combination of phonetics and cluster-based methods for spell checking was to our knowledge not yet explored and it is the new contribution of our work. We report an improvement of 8.16% accuracy when compared to a previously proposed spell checking approach.

Keywords: spell checking, clustering, phonetic algorithm

1 Introduction

Spelling English words correctly can be a rather daunting task to children and adults alike. English spelling is particularly difficult being very much unlike other languages, particularly those with a recent written form, in which the spelling of a word is directly related to the way it is pronounced. Mitton [1] discusses the pros and cons of the current English spelling system through a historical perspective as the English language changed in different directions: foreign words were incorporated, pronunciation patterns changed and new words were created. The orthography may either change accordingly to reflect those changes and make spelling easier and often simpler or it may conserve the old forms which inevitably creates difficulties to writers. Over the centuries, several attempts have been made to reform the English orthography and bring it closer to the current pronunciation of words, but there are still a great number of exceptions and difficulties in the English spelling system that writers must cope with.

Various surveys have shown that correct spelling is a difficult task even for those that are native speakers [1, 2]. Gorman [3] reported that seventy-two percent of adults, participating in a literacy scheme, seemed to have particular difficulty with spelling . In such scenario, one can safely assume that matters are even worst among non-native speakers, particularly those exposed to the English language for the first time during their adult life.

In order to provide help to such large population, word processors and even other types of applications such as browsers, tend to have some form of spel checking built in. The task of the latter is well-known in computational linguistics, its origins can be traced back to the work of Blair [4] and Damerau [5]. These spell checkers have two main functions, explained below.

First, to identify possible misspellings committed by users. Probably the easiest, and the most common, way to establish whether a given word is misspelled is by checking if it belongs to a list of correctly spelled words, a dictionary. Of course this presents difficulties, dictionaries tend to be rather large making this task computationally demanding. Another problem is that the misspelling may match a correct word that was not intended by the user, this is sometimes referred as *real-world* errors.

Second, to suggest the users' target word or a list of candidates containing the target word. In the case of the latter, a spell checker should aim for a short list, preferably ordered with the most likely intended spelling on the top. This list of possible target words produced by a spell checker can be seen as a ranked cluster of words with a degree of similarity to the misspelled word.

Clustering algorithms aim to group a set of N entities $y_i \in Y$, each described over V features, into K homogeneous groups $S = \{S_1, S_2, ..., S_K\}$. There are indeed a number of such algorithms, following partitional and hierarchical approaches. Hierarchical algorithms form a clustering tree in which a given entity y_i may belong to different clusters, as long as these are found at different levels in the hierarchy. Although useful these algorithms tend to have a high time complexity, in most cases of $\mathcal{O}(N^3)$. Partitional algorithms follow a different approach. They produce disjoint clusters in which an entity $y_i \in Y$ is assigned exclusively to a single cluster S_k. Such algorithms tend to have a much more favourable time complexity.

Clustering algorithms were recently applied to spell checking [6], proving to be an effective alternative to other spell checking methods, particularly those that rely on finite state automata (FSA) -based dictionaries. However, the success of the former strongly depends on the distance measure used to form clusters. The chosen distance has to accurately represent the similarity between the misspelling and the target word.

We find it very difficult to design a single distance measure that can handle all the different sources of misspellings, such as *typos* coming from neighbour letters in a keyboard, and those based on the difference between the sound and the spelling of a word. The former may be represented well by the Levenshtein metric [7], or one of its many variants. By using this metric, the distance between two words is given by the number of single character edits necessary to make them equal. Phonetic information, such as that produced by the Soundex algorithm [8], may provide us some clues regarding the target word as it may have a similar sound to that of the misspelling. Such information has been used in spell checking in different occasions such as the work of [9, 10] and [11]. Selecting an inappropriate measure may result in a large measured distance between a given misspelling and its respective target word, making it difficult for a clustering algorithm to suggest the correct target.

In this paper we aim to address the above issue, as well as the fact that measuring the distances between a given misspelling and each word in a dictionary can be quite computationally demanding. We do so by introducing a clustering-based method that allows the combination of different distance metrics. Our method improves the performance of spell checkers in terms of both, processing time and target word recovery. We believe this to be the first time that a clustering based spell checker is set to use phonetic as well as a string metric to form clusters of target words, given a misspelling.

2 Related Work

Spell checking has been substantially studied over the years. The aforementioned studies published by Blair [4] and Damerau [5] were, according to Mitton [12], the pioneers for this task. The majority of spell checking methods (including the one we present) use dictionaries as a list of correct spellings (target words). A few attempts, however, try to address this problem without the use of dictionaries. Morris and Cherry [13] was the first to use the frequency of character trigrams to calculate an 'index of peculiarity'. This coefficient estimates the probability of a given trigram occurring in English words. If a word contains a rare trigram the algorithm considers this word a good candidate for misspelling.

Some years later, McIlroy [14] introduced a technique called affix-stripping. This algorithm stores a stem word, e.g. *walk*, instead of all its possible derivations: *walking, walks, walked* and apply a set of rules to handle affixes. The method was effective in identifying misspellings but a shortcoming of this technique is the generation of non-existing words.

As previously mentioned, most state of the art spell checkers use dictionaries and similarity measures to calculate the distance between the target word and the possible misspellings [15]. This approach, however, tends to be computationally demanding when working with large dictionaries (the bigger the dictionary, the greater the number of distances to be calculated). There are two alternatives that have been propose to cope with this limitation, the first of them is the use of dictionaries organized as Finite State Automata (FSA) such as in Pirinen and Linden [16] and Pirinen [17] and the second is partitioning the dictionary is smaller clusters [6]. The latter is the strategy that will be used in our experiments, which is discussed in more details in Section 3.

A difficulty that most dictionary-based systems face is recognizing the so-called *real-word errors*, which are misspellings that generate words that exist in the dictionary. To identify *real-world errors* spell checkers must take context into account. For example, the word *haze* exists in English and will certainly be part of every dictionary and therefore not generate any alert when it is used in a text. However, when the sentence written is *I haze you* the distribution can tell the spell checkers that there is something wrong with the sentence, turning *haze* in this particular context in a good candidate of a misspelled word.

Confusion sets [18–20] are an interesting technique to cope with *real-world errors*. They are small groups of words that are often confused with one another, e.g. *(there, their, they're)* or *(we're, were)* or *(than, then, them)*. There are a number of methods developed to address the question of *real-word errors* such as Verberne [21], which proposed a probabilistic context-sensitive word trigram-based method. The method works under the assumption that the misspelling of a word often results in an unlikely sequence of three words.

Among the most recent studies, Islam and Inkpen [22] uses the Google Web IT 3-gram dataset to improve recall, Xue et al. [23] using syntactic and distributional information and more recently Lin and Chu [24] using N-gram models, similar character replacement, and filtering rules for Chinese.

3 Methods

There are indeed many clustering algorithm, generally divided into partitional and hierarchical. The latter have a considerably higher time complexity than the former, hence we focus on partitional clustering algorithms.

K-Means [25] is probably the most well-known partitional clustering algorithm there is. It partitions a data set $Y = \{y_1, y_2, ..., y_N\}$ into K disjoint partitions $S = \{S_1, S_2, ..., S_K\}$, each represented by a single centroid in $C = \{c_1, c_2, ..., c_K\}$. The centroid c_k of a given cluster S_k is the center of $y_i \in S_k$. In most cases K-Means is used in conjunction with the squared Euclidean distance, defined for the V-dimensional x and z as $d(x, z) = \sum_{v \in V}(x_v - z_v)^2$. In this case the definition of centroid is rather straight forward $c_{kv} = \frac{1}{|S_k|}\sum_{y_i \in S_k} y_{iv}$. Unfortunately finding the minimization of other distance measures, such as the Levenshtein distance, is not always possible. One should note centroids were designed to be synthetic representations of clusters, in other words, in most cases $c_k \notin Y$. This means that even if we were to somehow minimize the Levenshtein distance, we would be potentially representing clusters of words by a non-existing words.

The Partition Around Medoids (PAM) [26] addresses the above problem by replacing each centroid by a medoid. The medoid m_k of a given cluster S_k is equivalent to the entity $y_i \in S_k$ with the minimum sum of distances to all other $y_j \in S_k$. PAM minimises the criterion below.

$$W(S, M) = \sum_{k=1}^{K} \sum_{i \in S_k} \sum_{v \in V} (y_{iv} - m_{kv})^2, \tag{1}$$

where M represents a set of medoids $m_1, m_2, ..., m_K$. The criterion above represents the sum of distances between each medoid $m_k \in M$ and each entity $y_i \in S_k$, for $k = 1, 2, ..., K$. This Criterion is based on the squared Euclidean distance, but clearly we can substitute $(y_{iv} - m_{kv})^2$ for $d(y_{iv}, m_{kv})$, a function returning the distance between y_{iv} and m_{kv}. PAM allow us to use a distance measure whose center is unknown. We can minimize (1) with three simple steps.

1. Randomly select K entities from Y as initial medoids.
2. Assign each entity $y_i \in Y$ to the cluster formed by its closest medoid.
3. Update each medoid $m_k \in M$ to be equivalent to the entity $y_i \in S_k$ with the minimum sum of distances to all other $y_j \in S_k$, for $k = 1, 2, ..., K$.

The partitions produced by PAM are heavily influenced by the initial medoids found ramdonly in step one. There are of course many solutions for this problem. One could run PAM a number of times and choose as final partition the one that produces the smallest $W(S, M)$ (1), but this would have an impact on the amount of time taken to produce S. Another common solution is to apply a initialization algorithm so that the medoids are not simply found at random, but instead represent the final medoids, to some level. Regarding the second solution there are a various algorithms one could you, here we choose a variation of the original intelligent K-Means (iK-Means) [27], mainly because of our previous success in clustering dictionaries with it.[6].

The iK-Means algorithm was originally designed to find the number of clusters in a data set Y, as well as representative initial centroids for K-Means. This initialization does so by extracting one anomalous cluster from Y at a time, and initializes the K-Means centroids with those of the anomalous clusters. We have adapted iK-Means to work with medoids, rather than centroids, so it can initialize the PAM algorithm. We refer to our modification as the intelligent PAM (iPAM) algorithm, it alternatingly minimizes the below.

$$W(S_A, m_t) = \sum_{y_i \in S_A} d(y_i, m_t) + \sum_{y_i \in S_A} d(y_i, m_c), \tag{2}$$

where S_A is an anomalous cluster in Y, m_c is the entity $y_i \in Y$ with the smallest sum of distances to all other entities $y_j \in Y$, and m_t is the entity $y_i \in S_A$ with the smallest sum of distances to all other entities $y_j \in S_A$, we can see m_t as a tentative medoid based on S_A, and m_c as the medoid of the data set Y. Below, the algorithm for this minimization.

1. Set m_c as the entity with the smallest sum of distances to all other entities in the dataset Y.
2. Set m_t to the entity farthest away from m_c.
3. Apply PAM to Y using m_c and m_t as initial medoids, m_c should remain unchanged during the clustering.
4. Add m_t to M.
5. Remove m_t, and each $y_i \in S_A$ from Y. If there are still entities to be clustered go to Step 2.
6. Apply PAM to the original dataset Y, initialized by the medoids in M and $K = |M|$.

In order to complete (1) and (2) we need to define d, the function returning the distance between two strings. Using single distance measure would probably bias the clustering towards a single type of error. For instance, if we were to simply use the Levenshtein distance each cluster would contain words that are close to each other in terms of character insertions, deletions and substitutions. However, these clusters would not take into account phonetic similarities between words. We have decided to create a framework allowing the use of more than a single distance.

Our framework follows the idea that multiple distances can be combined, so that the distance between two strings x and z is given by

$$d(x,z) = \sum_{t=1}^{T} w_t d_t(x,z) \tag{3}$$

where w_t is the weight of distance d_t, allowing different distances to have different degrees of relevance for our particular task, and T is the total number of distances used. Equation (3) is subject to $\sum_{t=1}^{T} w_t = 1$. In the experiments for this paper we have decided to use only two distances, as per below.

$$d(x,z) = w_1 * Levenshtein(x,z) + w_2 * Levenshtein(Soundex(x), Soundex(z)), \tag{4}$$

where *Levenshtein* is a function returning the Levenshtein distance between two strings, and *Soundex* is a function returning the four digits Soundex code of a single string. This Soundex code is the same for homophones, allowing the matching of two strings that only have minor spelling differences. In our experiments we have set w_1 and w_2 to 0.5. Using (1), (2), and (4) we have developed a method used to find the target words of a misspellings. Our method is open to the use of virtually any quantity of distance measure, as long as they are valid for strings.

1. Apply the iPAM initialization to the dictionary using (4) to find the number of clusters K and a set of initial medoids M_{init}
2. Using the medoids in M_{init}, apply PAM with (4) to the dictionary to find K clusters. This should output a final set of medoids $M = \{m_1, m_2, ..., m_K\}$.
3. Given a misspelling z, calculate its distance (4) to each medoid $m_k \in M$. Save in M_* the medoids that have the distance to z equal to the minimum found plus a constant c.
4. Calculate the distance between z and each word in the clusters represented by the medoids in M_*, outputting the words whose distance is the minimum possible to z.
5. Should there be any more misspellings, go back to Step 3.

We have added a constant c to avoid our method getting trapped in local minima, increasing the chances of the algorithm finding the target word. Clearly a large c will mean more distance calculations. In our experiments we have used $c = 1$. By calculating distances initially between the misspelling z and each of the medoids $m_k \in M$ in Step 3 we reduce considerably the number of distance calculations needed to output our cluster of words. Instead of calculating 57,046 distances (the size of our dictionary, as described in Section 4), we reduce this number to $|M| + |S_{M_*}|$. In Section 5.1 we empirically show the latter to be much smaller than the former.

4 Experimental Setting

In order to perform our experiments we first acquired an English dictionary containing 57,046 words. This dictionary is used as our list of correctly spelled words. We have also downloaded the Birkbeck spelling error corpus[1], consisting of a list of 36,133 misspellings of 6,136 target words. This corpus has been used by various publications, including [28, 6].

The corpus includes misspellings from young children and extremely poor spellers who took part in spelling tests way beyond their ability. For this reason, some of the misspellings are completely different from their target words. As stated in the corpus description, the misspellings compiled were often very distant from the target words, examples include the misspellings *o*, *a*, *cart* and *sutl* for the targets *accordingly*, *above*, *sure* and *suitable*, respectively. As a second step, we removed from the corpus all misspellings whose targets were not present in the dictionary. These words would be impossible to be handled in a dictionary-based approach such as ours, thus there was no reason to keep them in the dictionary. This reduced the corpus to 34,956 misspellings, just under 97% of the original dataset.

Our method requires the clustering of our dictionary. This clustering has to be done only once, afterwards one can use our method as many times as required. However, this can be a time consuming experience as our dictionary is rather large. In order to reduce this processing time we divided the dictionary into 26 sub-datasets, based on the first letter of each word. Subsequently, we have then applied the first and second steps of our method to each of these 26 sub-datasets. This segmentation, however, does not mean that our method will not find the target word when the misspelling happens in the first letter.

5 Results

We are interested in evaluating our method in terms of processing time and target word recovery. Calculating the processing time of a particular algorithm can be a delicate matter. Computers using different hardware, and even different operating systems, may output different times for the same algorithm. An important point not to be overlooked is that one needs to be careful to measure the algorithm, and not the skills of a particular programmer. With these points in mind we have decided that a better processing time will be given by an algorithm that requires less distance calculations. Since our dictionary has 57,046 words, this is the maximum number of calculations needed to find a set of words in a dictionary that is similar to a particular misspelling.

[1] `http://www.dcs.bbk.ac.uk/roger/corpora.html`

Regarding target word recovery, there are two measures that we need to take into account. First, given a misspelling z, our method returns a cluster S_k containing candidate words. Our method should include the target word of z in S_k. Second, we aim to have a rather small $|S_k|$, preferably $|S_k| = 1$, since S_k would be meaningless to most users if for example $|S_k| = 100$.

Table 1 presents the results of our main experiments. We take as baseline the previous research carried out by de Amorim and Zampieri (2013) [6], which used a similar clustering algorithm but did not include the use of phonetic information. It is important to point out that in [6] researchers reported results in terms of success rate. They considered as success: 1) cases in which the method returned the correct target word 2) cases in which the method returned the word with the smallest Levenshtein distance as the target word. In this paper we consider accuracy to be a more interesting measure than the above defined success rate. This allows us to have an estimation of how suitable is our method to real-world applications and for this reasons we take only the correct target words into account.

We can clearly see that by including the phonetic information in the distance calculation (4) our method has improved considerably. The new version we present in this paper is able to recover the correct target word of 2,146 more misspellings than if phonetic information was not used. We can also see that $|S_k|$, the cluster of candidate words, has decreased considerably in terms of mean size, as well as median. Given a misspelling z our method presents, in average, a cluster of potential targets S_k whose cardinality is of 2.17. In fact, in 59.72% of cases, our method suggests a single candidate word.

The number of distance calculations has also decreased considerably. For a given misspelling z, our method calculates its distance, in average, to 3,175.3 words in the dictionary. This number represents 5.57% of the total number of words in our dictionary.

Method	Clustering	Clustering + Phonetics
Accuracy (%)	41.71%	47.85%
Accuracy (Nominal)	14,579 words	16,725
Total Number of Clusters	1,570 clusters	1,215
Cluster Size (Mean)	3.78 words	2.17 words
Cluster Size (Median)	2 words	1 word
Average Distance Calculations	3,251.4	3,175.3

Table 1. Results: The use phonetic information in comparison to those presented in de Amorim and Zampieri (2013)

We find that our method compares favourably to others in the literature. Mitton[28] presents various experiments using the Birkbeck spelling error corpus. In one set of experiments variations were generated of a given misspelling z, if this variation belonged to the dictionary it would be kept as a potential target word. This method meant that for each variation of z one would need to verify if it was equivalent to each of the words in the dictionary Clearly this generates a considerable computational effort, by generating these variations and checking each one of them against a dictionary. The success rate of this method was of 33%, however, it did not produce a single candidate word to 44% of the misspellings.

Mitton [28] also experimented by assembling a collection of potential target words by representing words and misspellings with a phonetic key originated from the SPEEDCOP project [29]. Under this approach the success rate improved radically to 74.2%, however, each cluster had an average of 600 candidate words, hardly realistic. The number of distance calculations used in this approach would be equal to the size of the dictionary for any given misspelling.

The cluster of candidate words generated by the above method can be ranked. Mitton [28] experimented a ranking based on letter by letter matching between the misspelling and the candidate words. The correct target word was found among the top three ranked candidates in 47.4% of cases, however, the number of distance calculations increases considerably. With an average cluster of 600 words this means a total number of calculations, just for the ranking part of this method, equivalent to the number of misspellings times 600. This is over 20 million distance calculations for our data set. Instead of matching letters one could use the edit distance, or even other methods suggested by Mitton [28], which do indeed increase the success rate, however, these do not seem to drastically decrease the required number of distance calculations to the level of our method.

5.1 Parameter Tuning: The Constant c

Our method uses a constant c in order to reduce the chances of our method getting trapped in local optima (see Section 3 for details). Originally we experimented with $c = 1$, by increasing this constant one can expect to increase the accuracy of our method, at the cost of increasing the number of distance calculations as well. Table 2 presents our experiments with $c = 2$ and $c = 3$.

Constant c Value	$c = 2$	$c = 3$
Accuracy (%)	50.44%	50.59%
Accuracy (Nominal)	17,631 words	17,270 words
Total Number of Clusters	1,570 clusters	1,215 clusters
Cluster Size (Mean)	2.23 words	2.24 words
Cluster Size (Median)	1 word	1 word
Average Distance Calculations	13,573	35,450

Table 2. Results: Tuning the constant c.

We obtained the best performance for our method when using $c = 3$. Along with the performance improvement we can see that the number of distance calculations increases rather rapidly, which makes the method slower than when using the constant $c = 1$. Therefore, for this setting, we do not believe this variable should be fine tuned.

6 Conclusion

In this paper we have introduced the use of phonetic information to clustering based spell checking methods. We do so by proposing a method that allows the combination of the distance bias of different distance measures. Here, we have experimented with the popular Levenshtein distance to find the distance between two strings in terms of character insertion, deletion and substitution, as well as the Levenshtein distance between the Soundex code of the the strings in order to detect the phonetic similarity between them.

The success of our method can be measured in different ways. Given a misspelling, our method produces a cluster of potential target words whose cardinality is smaller than that of previous research. That means that the list of suggestions given to a user has in average only 2.17 words with a median of 1 - for most misspellings our method suggested a single target word. We can also see a considerable improvement in terms of having the correct target word in the cluster, or one with a distance to the misspelling that is smaller than that of the target word. Finally, we have reduced the number of distance calculations from 57,046 (if no clustering is used), which is the cardinality of our dictionary, to an average of 3,175.3, about 5.57% of the original number.

We find it important to point out that even if another method is used to reduce the number of distance calculations, for instance by assuming that the first letter of the misspelling is correct, one can still reduce calculations further by implementing our method as a second stage to that.

6.1 Future Work

We still see considerable room for improvement. At the present we treat each distance equally by setting each w in (4) to 0.5. In future research we intend to investigate the possibility of designing an adaptive weighting system that would optimize each value of w to the best values for a particular user. To carry out such training there is unfortunately a lack of language resources available. Research in spell checking depends heavily on the availability of spelling error corpora and to our knowledge very few suitable resources are available.

In future work we would like to investigate whether performance can be increased by using other phonetic algorithms such as the Metaphone [30] or Double Metaphone [31] as well as by using other string distance functions [5]. Another direction that our work will take is the application of this clustering method to similar problems, particularly those in which phonetic information plays an important role. The method can be applied, for example, to carry out spelling normalization in internet texts [32] or historical data [33].

References

1. Mitton, R.: English spelling and the computer. Longman (1996)
2. Hamilton, M., Stasinopoulos, M.: Literacy, numeracy and adults: Evidence from the national child development study (1987)
3. Gorman, T.P.: A survey of attainment and progress of learners in adult literacy schemes. Educational Research **23**(3) (1981) 190–198
4. Blair, C.: A program for correcting spelling errors. Information and Control **3** (1960) 60–67
5. Damerau, F.: A technique for computer detection and correction of spelling errors. Communications of the ACM **7** (1964) 171–176
6. de Amorim, R., Zampieri, M.: Effective spell checking methods using clustering algorithms. In: Proceedings of Recent Advances in Natural Language Processing (RANLP2013), Hissar, Bulgaria (2013) 172–178
7. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. In: Soviet physics doklady. Volume 10. (1966) 707
8. Russel, R.: Soundex (1981)
9. Uzzaman, N., Khan, M.: A bengali phonetic encoding for better spelling suggestions. In: Proceeding of the 7th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladsh (2004)
10. Stüker, S., Fay, J., Berkling, K.: Towards context-dependent phonetic spelling error correction in children's freely composed text for diagnostic and pedagogical purposes. In: INTERSPEECH. (2011) 1601–1604
11. Toutanova, K., Moore, R.C.: Pronunciation modeling for improved spelling correction. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics (2002) 144–151
12. Mitton, R.: Fifty years of spellchecking. Writing Systems Research **2** (2010) 1–7
13. Morris, R., Cherry, L.: Computer detection of typographical errors. IEEE Transactions on Professional Communication **18** (1975) 54–64
14. McIlroy, M.: Development of a spelling list. IEEE Transactions on Communications **1** (1982) 91–99
15. de Amorim, R.: An adaptive spell checker based on ps3m: Improving the clusters of replacement words. Computer Recognition Systems 3 (2009) 519–526
16. Pirinen, T., Linden, K.: Finite-state spell-checking with weighted language and error models. In: Proceedings of the Seventh SaLTMiL workshop on creation and use of basic lexical resources for less-resourced languages, Malta (2010)
17. Pirinen, T.: Weighted Finite-State Methods for Spell-Checking and Correction. PhD thesis, University of Helsinki (2014)
18. Golding, A., Roth, D.: A winnow-based approach to context-sensitive spelling correction. Machine Learning **34** (1999) 107–130
19. Carlson, A., Rosen, J., Roth, D.: Scaling up context-sensitive text correction. In: Proceedings of the 13th Innovative Applications of Artificial Intelligence Conference, AAAI Press (2001) 45–50
20. Pedler, J., Mitton, R.: A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. In: Proceedings of LREC2010, Malta (2010)
21. Verberne, S.: Context-sensitive spell checking based on word trigram probabilities. Master's thesis, University of Nijmegen (2002)
22. Islam, A., Inkpen, D.: Real-word spelling correction using googleweb 1t 3-grams. In: Proceedings of Empirical Methods in Natural Language Processing (EMNLP2009), Singapore (2009) 1241–1249

23. Xu, W., Tetreault, J., Chodorow, M., Grishman, R., Zhao, L.: Exploiting syntactic and distributional information for spelling correction withweb-scale n-gram models. In: Proceedings of Empirical Methods in Natural Language Processing (EMNLP2011), Edinburgh, Scotland (2011) 1291–1300
24. Lin, C., Chu, W.: Ntou chinese spelling check system in sighan bake-off 2013. In: Proceedings of the Seventh SIGHAN Workshop on Chinese Language Processing (SIGHAN-7), Nagoya, Japan (2013) 102–107
25. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. Volume 1., California, USA (1967) 14
26. Kaufman, L., Rousseeuw, P.: Finding groups in data: an introduction to cluster analysis. Volume 39. Wiley Online Library (1990)
27. Mirkin, B.: Clustering for data mining: a data recovery approach. Volume 3. CRC Press (2005)
28. Mitton, R.: Ordering the suggestions of a spellchecker without using context. Natural Language Engineering **15** (2009) 173–192
29. Pollock, J.J., Zamora, A.: Automatic spelling correction in scientific and scholarly text. Communications of the ACM **27**(4) (1984) 358–368
30. Philips, L.: Hanging on the metaphone. Computer Language **7**(12 (December)) (1990)
31. Philips, L.: The double metaphone search algorithm. C/C++ users journal **18**(6) (2000) 38–43
32. Zampieri, M., Hermes, J., Schwiebert, S.: Identification of patterns and document ranking of internet texts: A frequency-based approach. In: ZSM Studien, Special Volume on Non-Standard Data Sources in Corpus-Based Research. Volume 5. Shaker (2013)
33. Baron, A., Rayson, P.: Vard2: A tool for dealing with spelling variation in historical corpora. In: Postgraduate conference in corpus linguistics. (2008)