Effective Spell Checking Methods Using Clustering Algorithms

Renato Cordeiro de Amorim
Glyndŵr University, UK
r.amorim@glyndwr.ac.uk

Marcos Zampieri
University of Cologne, Germany
mzampier@uni-koeln.de

Abstract

This paper presents a novel approach to spell checking using dictionary clustering. The main goal is to reduce the number of times distances have to be calculated when finding target words for misspellings. The method is unsupervised and combines the application of anomalous pattern initialization and partition around medoids (PAM). To evaluate the method, we used an English misspelling list compiled using real examples extracted from the Birkbeck spelling error corpus.

1 Introduction

Spell checking is a well-known task in computational linguistics, dating back to the 1960s, most notably to the work of Damerau (1964). Nowadays, spell checkers are an important component of a number of computer software such as web browsers, text processors and others.

In recent years, spell checking has become a very important application to search engines (Martins and Silva, 2004). Companies like *Google* or *Yahoo!* use log files of all users' queries to map the relation between misspellings and the intended spelling reaching very high accuracy. The language of queries, however, is typically shorter than naturally occurring text, making this application of spell checking very specific (Whitelaw et al., 2009).

Spell checking methods have two main functions. The first one is to identify possible misspellings that a user may commit. As described by Mitton (1996), misspellings can be related to the writer's (poor) writing and spelling competence, to learning disabilities such as dyslexia, and also to simple performance errors, known as *typos*. The written production of non-native speakers also plays an important role in spell checking as they are, on average, more prone to errors than native speakers. These phenomena generate a wide range of different spelling possibilities that a spell checker should be trained to recognize.

The second function of spell checkers is to suggest the users' intended spelling of a misspelled word or at least to suggest a list of candidates in which the target word appears. This is often done by calculating the distance between the misspelled word and a set of potential candidates. As will be discussed in this paper, this is by no means trivial and several methods have been proposed to address this task.

This paper presents a novel unsupervised spell checking method combining anomalous pattern initialization and partition around medoids (PAM). To the best of our knowledge this is the first attempt to apply these methods for spell checking. The approach described here aims to improve spell checking' speed and performance.

2 Related Work

Spell checking techniques have been substantially studied over the years. Mitton (2010) points out that the first attempt to solve the problem can be traced back to the work of Blair (1960) and later more attention was given to the work of Damerau (1964). Most spell checking methods described in the literature, including this one, use dictionaries as a list of correct spellings that help algorithms to find target words. Only a few attempts try to address this problem without the use of dictionaries (Morris and Cherry, 1975).

Morris and Cherry 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 trigram is rare in English, the algorithm flags the word containing this trigram as a misspelled one. For example, *wha* is a frequent trigram in English whereas *wah* is not, therefore the word *waht* is very likely to be assigned as a misspelling by the

Proceedings of Recent Advances in Natural Language Processing, pages 172–178,
Hissar, Bulgaria, 7-13 September 2013.

system.

In the 1970s, the main issue with dictionary-based approaches was computing power. The small size of computer memories was a bottleneck for this kind of approach, as systems should ideally hold all entries of the dictionary in memory. The solution was to keep the dictionary on disk and retrieve small portions of it, storing them in the main memory when required. This was extremely time consuming. One technique used to minimize this limitation was to use affix-stripping (McIlroy, 1982). The basic idea is to store a stem word, e.g. *read*, instead of all its possible derivations: *reading, readable, reads, etc.* and apply a set of rules to handle affixes and adjust the stems if necessary. The method proved to be effective in identifying misspellings but it failed to suggest suitable target words, as in this process non-existent words were often generated such as *unreading* or *readation*.

In the present day, the challenge of coping with short memory size no longer exists. It is possible to store large-sized dictionaries in memory for immediate processing without using the disk to store data. However, dictionary-based techniques (de Amorim, 2009), still have a performance limitation due to their intrinsic architecture. State-of-the-art spell checking techniques often apply similarity metrics to calculate the distance between the target word and possible candidates in the dictionary. The bigger the dictionary, the greater the number of calculations, making the algorithms' performance slower. One common alternative to this performance limitation is the use of dictionaries organized as Finite State Automata (FSA) such as in Pirinen and Linden (2010b). These techniques will be better explained in section 2.1.

2.1 State-of-the-art Approaches

A known shortcoming of dictionary-based systems is handling so-called *real-word errors*. This kind of error is difficult to identify using these methods because the misspelled word exists in the dictionary. It is only by taking context into account that these misspellings become recognizable, such as in *better then me* or *were the winners*. The use of confusion sets (Golding and Roth, 1999; Carlson et al., 2001) is a solution to this problem. Confusion sets are a small group of words that are likely to be confused with one another, e.g. *(there, their, theyre)* or *(we're, were)* or *(than, then, them)*. The use of confusion sets in spell checking approaches takes syntax and semantics into account.

A number of confusion sets are provided to the spell checker, so that the context (words in window size n) in which a given target word occurs can be used to assess if the target word was correctly written or not. Carlson et al. (2001) uses 265 confusion sets and later Pedler and Mitton (2010) increases this number to 6,000 confusion sets reporting around 70% of real-word errors detected. Another approach to tackle *real-word errors* is the one by Verberne (2002) which proposed a context-sensitive word trigram-based method calculated using probability. The method works under the assumption that the misspelling of a word often results in an unlikely sequence of (three) words. To calculate this probability, the method uses the British National Corpus (BNC) as training corpus.

Other spell checking methods developed to address the question of real-word errors include the one by Islam and Inkpen (2009). This method uses the Google Web IT 3-gram dataset and aims to improve recall rather than precision. It reports 0.89 recall for detection and 0.76 recall for correction outperforming two other methods for the same task. More recently, Xue et al. (2011) address this problem using syntactic and distributional information.

The vast majority of state-of-the-art spell checking systems use similarity measures to compare the distance between two strings (Damerau, 1964; Levenshtein, 1966). Algorithms consider words that are not found in the dictionary as misspelling candidates. The distance between the candidates or target words to all words in the dictionary is then calculated and the words with the smallest distance are presented as suggestions. Using these techniques, spell checkers have become very effective at offering the top candidates of these suggestions lists as the correct spelling, creating what is described in the literature as the Cupertino Effect[1].

Another important aspect of state-of-the-art spell checkers is the aforementioned organization

[1] The Cupertino Effect was named after an anecdotal yet representative spell checking problem of the 1990s. Microsoft Word did not have the spelling *cooperation* in its dictionary, but the hyphenated one: *co-operation*. When someone typed *cooperation*, the system would offer *Cupertino* as its first suggestion.

of dictionaries as Finite State Automata (FSA). FSA-based methods use techniques from finite state morphology (Beesley and Karttunen, 2003) where the finite set of states of a given automaton correspond to characters of the words in the dictionary. FSA are particularly interesting for morphologically rich languages such as Finnish, Hungarian and Turkish. One example of a resource for spell checking that organizes the dictionary as FSA is Hunspell[2] originally developed for Hungarian, but adapted to several other languages (Pirinen and Linden, 2010a).

The technique presented in this paper serves as an alternative to the FSA-based dictionaries that reduce the number of distances that have to be calculated for each misspelling and therefore improving processing speed. Hulden (2009) observes that the calculation of distances is time consuming and investigates techniques to find approximate string matches in FSA faster. He defines the problem as 'a single word w and a large set of words W, quickly deciding which of the words in W most closely resembles w measured by some metric of similarity, such as minimum edit distance' and points out that finding the closest match between w and a large list of words, is an extremely demanding task.

3 Anomalous Pattern Initialization and PAM

The partition around medoids (PAM) algorithm (Kaufman and Rousseeuw, 1990) divides a dataset Y into K clusters $S = \{S_1, S_2, ..., S_K\}$. Each cluster S_k is represented by a medoid m_k. The latter is the entity $y_i \in S_k$ with the smallest distance to all other entities assigned to the same cluster. PAM creates compact clusters by iteratively minimising the criterion below.

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

where V represents the features of the dataset, and M the returned set of medoids $\{m_1, m_2, ..., m_K\}$. This criterion represents the sum of distances between each medoid m_k and each entity $y_i \in S_k$. The minimisation of (1) follows the algorithm below.

1. Select K medoids at random from Y, $M = \{m_1, m_2, ..., m_K\}$, $S \leftarrow \emptyset$.

2. Update S by assigning each entity $y_i \in Y$ to the cluster S_k represented by the closest medoid to y_i. If this update does not generate any changes in S, stop, output S and M.

3. Update each medoid m_k to the entity $y_i \in S_k$ that has the smallest sum of distances to all other entities in the same cluster. Go back to Step 2.

PAM is a very popular clustering algorithm and it has been used in various scenarios. However, it does have known weaknesses, for instance: (i) its final clustering depends heavily on the initial medoids used, and these are normally found at random; (ii) it requires the user to know how many clusters there are in the dataset; (iii) because of its iterative nature, it may get trapped in local optima; (iv) it does not take into account different features that may have varying degrees of relevance.

Weakness (iv) has been the subject of our previous research in feature weighting using cluster dependent weights and the L$_p$ norm (de Amorim and Fenner, 2012). We do not deal with this nor weakness (iii) in this paper, leaving them for future research in our particular scenario. Here we do address the intrinsically-related weaknesses (i) and (ii). It is impossible to define good initial medoids for PAM without knowing how many of these should be used.

The above has lead to a considerable amount of research addressing the quantity and initial position of medoids. Such effort generated a number of algorithms addressing one or both sides of the problem, such as Build (Kaufman and Rousseeuw, 1990), anomalous pattern initialization (Mirkin, 2005), the Hartigan index (Hartigan and Wong, 1979) and other initializations based on hierarchical clustering (Milligan and Isaac, 1980).

There have been numerous comparisons of various initializations on different scenarios (Chiang and Mirkin, 2010; Emre Celebi et al., 2013; de Amorim, 2012; de Amorim and Komisarczuk, 2012), leading us to conclude that it is difficult to appoint a single initialization that would always work. However, we do see the anomalous pattern initialization introduced by Mirkin (2005) favourably. His initialization addresses both sides of the problem and researchers observed previous success using it (Chiang and Mirkin, 2010; de Amorim, 2012; de Amorim and Komisarczuk, 2012).

[2] http://hunspell.sf.net

This initialization was originally designed for K-Means, taking the name intelligent K-Means. Below we present our medoid version of the anomalous pattern initialization, which we have used in our experiments.

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 its cluster 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 $K = |M|$.

Based on the above we have developed a method used to find the target words of misspellings. Our method is open to the use of virtually any distance measure valid for strings. Our main aim with this method is to reduce the number of times distances have to be calculated. To do this we apply the anomalous pattern initialization and PAM, as per below.

1. Apply the anomalous pattern initialization to the dictionary, finding the number of clusters K and a set of initial medoids M_{init}
2. Using the medoids in M_{init}, apply PAM 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 w, calculate its distance to each medoid $m_k \in M$. Save in M_* the medoids that have the distance to w equal to the minimum found plus a constant c.
4. Calculate the distance between w and each word in the clusters represented by the medoids in M_*, outputting the words whose distance is the minimum possible to w.
5. Should there be any more misspellings, go back to Step 3.

We have added a constant c to increase the chances of the algorithm finding the target word. Clearly a large c will mean more distance calculations. In our experiments with the Levenshtein distance (Levenshtein, 1966) we have used $c = 1$.

4 Setting of the Experiment

For our experiments we first acquired an English dictionary containing 57,046 words, and a corpus consisting of a list of 36,133 misspellings together with its 6,136 target words[3]. This misspelling list was previously used by Mitton (2009) and it was extracted from the Birkbeck spelling error corpus. The corpus includes misspellings from young children as well as extremely poor spellers subject to spelling tests way beyond their ability. For this reason, some of the misspellings are very different from their target words. As stated in the guidelines of the corpus, the misspellings compiled were often very distant from the target words, examples of these include the misspellings o, a, $cart$ and $sutl$ for the targets $accordingly$, $above$, $sure$ and $suitable$, respectively.

As a second step, we removed from our corpus all misspellings whose targets were not present in the dictionary. This reduced the corpus to 34,956 misspellings, just under 97% of the original dataset. Dictionaries tend to be large, making their clustering time consuming. In order to reduce this processing time we segmented the dictionary in 26 sub-datasets, based on the first letter of each word. 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. The clustering of a large dictionary can be time consuming. However, this needs to be done only once.

We took Peter Norvig's (2009) spell checker[4] as our baseline performance. This spell checker is a simplified adaptation of the methods used in *Google* and is being frequently used as baseline for state-of-the-art experiments in spell checking. For our method, Norvig's experiments are particularly interesting because it uses the same dataset, the Birkbeck Spelling Error Corpus. The author reports performance of 74% for a development dataset and 67% for a test dataset. To use as baseline we consider Norvig's best result, 74% success rate, plus 3.24%, which is the percentage of

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

[4] http://norvig.com/spell-correct.html

the dataset that we did not consider in our experiments. This results in a baseline performance of 77.24%.

The use of Norvig's method in this paper is exclusive to serve as a baseline performance and not an attempt to compare both methods. As it will be discussed next section, the two methods are conceptually different, making it very difficult to stablish a fair-ground comparison between them. We see Norvig's simplistic adaptation of *Google*'s algorithm for spell checking the same way as, for example, the majority class baseline is used in text classification. In other words, the minimum expectable performance that an algorithm should achieve.

5 Results

The main aim of our method is to reduce the number of times distances are calculated. Should one measure the distance between a misspelling and each word in our dictionary, this distance function would be called 57,046 times, the size of the dictionary. By applying our method to each of the 34,956 misspelling in the corpus we previously described, the distance measure was calculated on average 3,251.4 times for each misspelling. We find this is an important result from a computational point of view, as we are reducing considerably the number of calculations.

Regarding the recovery of the target words, it depends very much on the distance measure in use. We have experimented with the popular Levenshtein distance (Levenshtein, 1966). In 88.42% of cases our method returned a cluster containing the target word or a word with a smaller distance to the misspelling than the target. We attribute some of the latter to misspellings that are actual words (real-word errors), an issue that we do not address in this paper. Results are summarized in table number 1:

Total Misspellings	34,956 words
Success Rate (%)	88.42%
Success Rate (Nominal)	30,908 words
Baseline Gain (pp)	+ 11.18
Total Number of Clusters	1,570 clusters
Average Cluster Length	3.78 words
Average Distance Calculations	3,251.4

Table 1: Results

The cardinality of the clusters returned by our method is also of interest. Ideally the clusters should be rather small, so that users can easily identify the target word in the cluster. In our experiments with the corpus, the average cluster contained 3.78 words, with a median of 2. However, in 7.98% of cases the cluster had over 10 words.

We find the results obtained quite promising as the method outperforms the baseline in 11.18 percentage points[5] using the same dataset (this number takes into account that we had to reduce ours in just over 3%, as described in Section 4). As mentioned in section 4, the corpus contains many misspellings whose target we find impossible to identify.

As previously mentioned, there are a few factors we should take into account when considering Norvig's (2009) method as baseline. His method is based on supervised learning, requiring a rather large sample of misspellings and their corresponding targets - our method has no such requirement and it is open to the use of various distance measures. As an example, he states that his method achieves better performance when 'pretending that we have seen the correctly spelled word 1, 10, or more times'. Another different aspect of both methods is that his method returns a single suggested target, while ours returns a cluster of suggested target words.

6 Conclusion

The method we introduced in this paper reduces the number of distances to be calculated without removing a single word from the dictionary. This makes the algorithm faster than other approaches and presents a satisfactory success rate of 88.42% in a challenging dataset. The success rate is 11.18% higher than the baseline for this task. The question of using a supervised method as a baseline performance have also been discussed in this paper.

We decided to work with a large complete dictionary, in contrast to a number of studies that discard rare words to decrease the number of instances in the dictionary. This decision was based on previous studies (Damerau and Mays, 1989). As stated by Mitton (Mitton, 2010): 'when people use a rare word, it is very likely to be a correct spelling and not a real-word error'. Therefore, a spell checker with a small dictionary would

[5] As discussed in section 4, Norvig's method returns a candidate to the target word, while ours return a cluster. We consider the success rate score of 88.42% and this does not correspond to accuracy or precision.

be very likely to raise false alarms over correctly spelt rare words.

As previously mentioned, the corpus contained the attempts of very poor spellers and therefore misspelled words were often very far from their targets. Another shortcoming of the corpus is the fact that it is organized as a simple list of words without context, making it difficult to refine calculations specifically for real-word errors.

6.1 Future Work

We are continuing the experiments described here and taking them in a couple of directions. First we aim to experiment by reducing the cardinality of clusters and by ranking words in these clusters. In so doing, suggestions presented by the algorithm would be even more accurate and suitable for real-world applications. Another aspect we would like to explore is the use of measures that learn from a corpus of misspellings, such as the one presented by de Amorim (2009).

As previously mentioned, in terms of processing speed, we see our method as an alternative to FSA-based methods. We are at the moment comparing the performance of our algorithm to state-of-the-art FSA-based methods, trying to stablish fair metrics to compare our cluster-based unsupervised method to supervised FSA methods. Methods are conceptually different in their architectures and establishing a fair ground for comparison is by no means trivial.

We would also like to investigate the possibility of reducing the number of distance calculations even further by merging our method with finite state automata, using a dictionary containing solely stem words. Under this approach we would have a smaller amount of medoids, however, this could have a considerable impact on accuracy.

Finally, we aim to replicate these experiments to a corpus in which misspellings are present in running text. This would make it possible to use context to improve the calculation of distances with features commonly used in other NLP problems such as word sense disambiguation (Zampieri, 2012). In so doing, we believe the results obtained by our method would be improved.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable suggestions to increase the quality of this paper.

References

K. Beesley and L. Karttunen. 2003. Finite-state morphology. *CSLI.*

C. Blair. 1960. A program for correcting spelling errors. *Information and Control*, 3:60–67.

A.J. Carlson, J. Rosen, and D. Roth. 2001. Scaling up context-sensitive text correction. In *Proceedings of the 13th Innovative Applications of Artificial Intelligence Conference*, pages 45–50. AAAI Press.

M.M.T. Chiang and B. Mirkin. 2010. Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads. *Journal of classification*, 27(1):3–40.

F. Damerau and E. Mays. 1989. An examination of undetected typing errors. *Information Processing & Management*, 25(6):659–664.

F. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7:171–176.

R.C. de Amorim and T. Fenner. 2012. Weighting features for partition around medoids using the minkowski metric. *Lecture Notes in Computer Science*, 7619:35–44.

R.C. de Amorim and P. Komisarczuk. 2012. On initializations for the minkowski weighted k-means. *Lecture Notes in Computer Science*, 7619:45–55.

R.C. de Amorim. 2009. An adaptive spell checker based on ps3m: Improving the clusters of replacement words. *Computer Recognition Systems 3*, pages 519–526.

R.C. de Amorim. 2012. An empirical evaluation of different initializations on the number of k-means iterations. *Lecture Notes in Computer Science*, 7629:15–26.

M. Emre Celebi, H.A. Kingravi, and P.A. Vela. 2013. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210.

A. Golding and D. Roth. 1999. A winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130.

J.A. Hartigan and M.A. Wong. 1979. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108.

M. Hulden. 2009. Fast approximate string matching with finite automata. *Procesamiento del Lenguaje Natural*, 43:57–64.

A. Islam and D. Inkpen. 2009. Real-word spelling correction using googleweb 1t 3-grams. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP2009)*, pages 1241–1249, Singapore.

L. Kaufman and P.J. Rousseeuw. 1990. *Finding groups in data: an introduction to cluster analysis*, volume 39. Wiley Online Library.

V. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*.

Bruno Martins and Mário J Silva. 2004. Spelling correction for search engine queries. In *Advances in Natural Language Processing*, pages 372–383. Springer.

M. McIlroy. 1982. Development of a spelling list. *IEEE Transactions on Communications*, 1:91–99.

G.W. Milligan and P.D. Isaac. 1980. The validation of four ultrametric clustering algorithms. *Pattern Recognition*, 12(2):41–50.

B. Mirkin. 2005. *Clustering for data mining: a data recovery approach*, volume 3. CRC Press.

R. Mitton. 1996. *English spelling and the computer*. Longman.

R. Mitton. 2009. Ordering the suggestions of a spellchecker without using context. *Natural Language Engineering*, 15(2):173–192.

R. Mitton. 2010. Fifty years of spellchecking. *Writing Systems Research*, 2(1):1–7.

R. Morris and L. Cherry. 1975. Computer detection of typographical errors. *IEEE Transactions on Professional Communication*, 18:54–64.

P. Norvig. 2009. Natural language corpus data. In *Beautiful Data*, pages 219–249. O'Reilly.

J. Pedler and R. Mitton. 2010. A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. In *Proceedings of LREC 2010*, Malta.

T. Pirinen and K. Linden. 2010a. Creating and weighting hunspell dictionaries as finite-state automata. *Investigationes Linguisticae*, 21.

T. Pirinen and K. Linden. 2010b. Finite-state spellchecking 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.

S. Verberne. 2002. Context-sensitive spell checking based on word trigram probabilities. Master's thesis, University of Nijmegen.

C. Whitelaw, B. Hutchinson, G. Chung, and G. Ellis. 2009. Using the web for language independent spellchecking and autocorrection. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP2009)*, pages 890–899, Singapore.

W. Xu, J. Tetreault, M. Chodorow, R. Grishman, and L. Zhao. 2011. Exploiting syntactic and distributional information for spelling correction withweb-scale n-gram models. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP2011)*, pages 1291–1300, Edinburgh, Scotland.

M. Zampieri. 2012. Evaluating knowledge-rich and knowledge-poor features in automatic classification: A case study in WSD. In *Proceedings of the 13th IEEE International Symposium on Computational Intelligence and Informatics (CINTI2012)*, pages 359–363, Budapest, Hungary.