White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of York.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/775/

Book Section:
Hodge, V.J. and Austin, J. (2001) A novel binary spell checker. In: Dorffner, G., Bischof, H. and Hornik, K., (eds.) Artificial neural networks : ICANN 2001 : International Conference, Vienna, Austria, August 21-25, 2001 : proceedings. Lecture Notes in Computer Science. Springer, Berlin, Germany, pp. 1199-1204.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose
university consortium
Universities of Leeds, Sheffield & York

White Rose Consortium ePrints Repository

http://eprints.whiterose.ac.uk/

This is an author produced version of a chapter published in **Artificial neural networks : ICANN 2001 : International Conference, Vienna, Austria, August 21-25, 2001 : proceedings.**

White Rose Repository URL for this paper:
http://eprints.whiterose.ac.uk/archive/00000775/

Citation for the published chapter

Hodge, V.J. and Austin, J. (2001) *A novel binary spell checker.* In: Dorffner, G. and Bischof, H. and Hornik, K., (eds). Artificial neural networks : ICANN 2001 : International Conference, Vienna, Austria, August 21-25, 2001 : proceedings. Lecture Notes in Computer Science (2130). Springer-Verlag, Berlin, Germany, pp. 1199-1204.

Citation for this chapter

To refer to the repository paper, the following format may be used:

Hodge, V.J. and Austin, J. (2001) *A novel binary spell checker.* Author manuscript available at: http://eprints.whiterose.ac.uk/archive/00000775/. [Accessed: *date*].

Published in final edited form as:

Hodge, V.J. and Austin, J. (2001) *A novel binary spell checker.* In: Dorffner, G. and Bischof, H. and Hornik, K., (eds). Artificial neural networks : ICANN 2001 : International Conference, Vienna, Austria, August 21-25, 2001 : proceedings. Lecture Notes in Computer Science (2130). Springer-Verlag, Berlin, Germany, pp. 1199-1204.

White Rose Consortium ePrints Repository
eprints@whiterose.ac.uk

A Novel Binary Spell Checker

Victoria J. Hodge[1] and Jim Austin[1]

University of York, UK
{vicky,austin}@cs.york.ac.uk

Abstract. In this paper we propose a simple, flexible and efficient hybrid spell checking methodology based upon phonetic matching, supervised learning and associative matching in the AURA neural system. We evaluate our approach against several benchmark spell-checking algorithms for recall accuracy. Our proposed hybrid methodology has the joint highest top 10 recall rate of the techniques evaluated. The method has a high *recall* rate and low computational cost.

1 Introduction

Errors, particularly spelling and typing errors are abundant in human generated electronic text. For example, Internet search engines are criticised for their inability to spell check the user's query. This would prevent wasted computational processing, prevent wasted user time and make any system more robust as spelling and typing errors can prevent the system identifying the required information.

We describe an interactive spell checker that identifies spelling errors and recommends alternative spellings. The spell checker uses a hybrid approach to overcome phonetic spelling errors and the four main forms of typing errors: insertion, deletion, substitution and transposition. We use a Soundex-type coding approach (see Kukich [4]) coupled with transformation rules to overcome phonetic spelling errors. We use an n-gram approach [5] to overcome the first two forms of typing error and integrate a Hamming Distance approach to overcome substitution and transposition errors. Our spell checker aims to high recall accuracy possibly at the expense of precision. However, the scoring allows us to rank the retrieved matches so we can limit the number of possibilities suggested to the user to the top 10 matches, giving both high recall and precision.

Some alternative spelling approaches include: Levenshtein Edit Distance (see [4]) which scores similarity by the number of transformations required to turn the misspelt word into each lexicon word but runs slowly; Agrep [7] [6] is based on Edit Distance but uses an approach optimised for speed, however Agrep has the lowest recall in our evaluation; Aspell [1] which integrates transformation rules and phonetic code generation; and, the two benchmark approaches MS Word 97 and MS Word 2000. We evaluate our approach against these alternatives for quality of retrieval both recall - the percentage of correct words retrieved

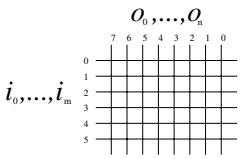

Fig. 1. The input vector i addresses the rows of the CMM and the output vector o addresses the columns.

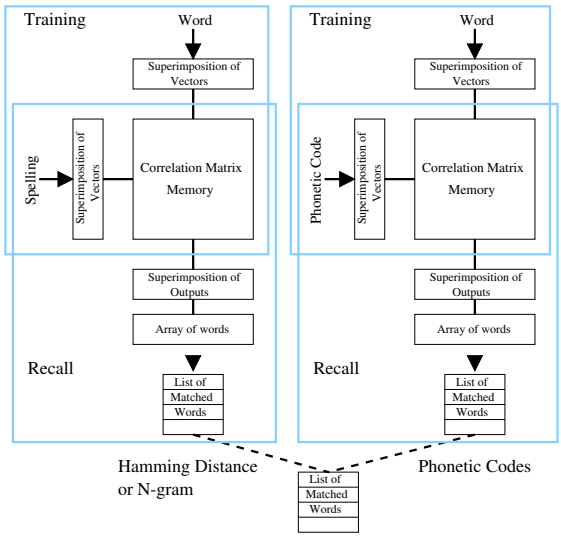

Fig. 2. Diagram of the hybrid spell checker in the AURA modular system.

from 600 misspelt words and precision - the number of false matches returned. The reader is referred to Kukich [4] for a thorough treatise of spell checking techniques.

1.1 Our Integrated Hybrid Modular Approach

AURA [2] is a collection of binary associative neural networks that may be implemented in a modular fashion. AURA utilises Correlation Matrix Memories (CMMs) to map inputs to outputs through a supervised learning rule, similar to a hash function see Fig. 1. In our system shown in Fig. 2, we use two CMMs [2]: one CMM stores the words for n-gram and Hamming Distance matching and the second CMM stores phonetic codes for homophone matching. The CMMs are used independently but the results are combined during the scoring phase of the spell checker to produce an overall score for each word.

Hamming Distance and N-Gram For Hamming Distance and n-gram, the word spellings form the inputs. We divide a binary vector of into a series of 30-bit chunks to allow 30 characters to be represented. Each word is divided into its constituent characters. The appropriate bit is set in the chunk to represent each character, in order of occurrence. The chunks are concatenated to produce a binary vector to represent the spelling of the word and form the input to the CMM. Any unused chunks are set to all zero bits. Each word in the lexicon has a unique binary vector to represent it with a single bit set corresponding to the word's position in the lexicon. The binary vector forms the output from the CMM for that word so we can identify when the word has been retrieved as a match.

Recalling from the Network - Hamming Distance Only the spelling input vector is applied to the network. The columns are summed, see (1).

$$output_j = \sum_{\text{all } i} input_i \wedge w_{ji} \tag{1}$$

We apply the Willshaw threshold which sets a bit in the output vector for each column summing to the threshold value or greater. We threshold at the highest summed column value to retrieve the best matches, i.e., words that match as many characters in the input as possible. The single binary vector output after thresholding is a superimposition of the best matching words' vectors.

Recalling from the Network - Shifting N-Grams We use exactly the same CMM for the n-gram method as we use for the Hamming Distance retrievals. However, we use 1-grams for spellings with less than 4 characters, 2-grams for 4 to 6 characters and 3-grams for spellings with more than 6 characters. Misspelt words with less than four characters are unlikely to have any 2-grams or 3-grams found in the correct spelling. Spellings with 4 to 6 characters may have no common 3-grams but should have common 2-grams and words with more than 6 characters should match 3-grams. For a 3-gram, we take the first three characters of the spelling, input these left aligned to the spelling CMM and threshold the output activation at the value three. We then slide the 3-gram one place to the right, input to the CMM and threshold at three. We continue sliding the 3-gram to the right until the first letter of the 3-gram is in the position of the last character of the spelling. We logically OR the output vector from each 3-gram position to produce an output vector denoting any word that has matched any of the 3-gram positions. We then move onto the second 3-gram, left align, input to the CMM, threshold and slide to the right producing a second 3-gram vector. When we have matched all n 3-grams from the spelling, we will have n output vectors representing the words that have matched each 3-gram respectively. We sum these output vectors to produce an integer vector representing a count of the number of 3-grams matched for each word. We then threshold at the maximum value to produce a thresholded binary vector with bits set corresponding to the best matching words.

Phonetic Spell Checking Our methodology combines Soundex-type codes with the phonetic transformation rules listed in Table 1 to produce a four-character code for each word. Any applicable transformation rules are first ap-

^hough → *h5*	^ps → s	1. c(e|i|y|h) → s 2. c → k
^cough → *k3*	^pt → t	gn$ → n
^chough → *s3*	^pn → n	gns$ → ns
^laugh → *l3*	^mn → n	1. (i|u)gh(¬a) → _ 2. gh → g
		mb$ → m
^rough → *r3*	^wr → r	ph → f
^tough → *t3*	^kn → n	q → k
^enough → *e83*	^gn → n	sc(e|i|y) → s
^trough → *tA3*	^x → z	+ti(a|o) → s
		+x → ks

Table 1. Table of the phonetic transformation rules in our system. Italicised letters are Soundex codes - all other letters are standard alphabetical letters. ^ indicates 'the beginning of a word', $ indicates 'the end of the word' and + indicates '1 or more letters'. Rule 1 is applied before rule 2.

plied to the word. The phonetic code for the word is then generated according to the algorithm in Fig. 3 using the codes given in Table 2. For the phonetic

01-2034004567809-ABC0D0-0B0000
abcdefghijklmnopqrstuvwxyz-'&/

Table 2. Table giving our codes for each letter.

vectors, we divide the vector into an initial alphabetical character representation (23 characters as c, q and x are mapped to other letters by transformation rules) and three 13-bit chunks. Each of the three 13-bit chunks represents a phonetic code from table 2 where the position of the bit set is the hexadecimal value of the code. Each word's output vector is identical to the Hamming Distance and n-gram CMM output vector for uniformity

Recalling from the Network - Phonetic Recall from the phonetic CMM is essentially similar to the Hamming Distance recall. We input the 4-character phonetic code for the search word into the CMM and recall a vector representing the superimposed outputs of the matching words. The Willshaw threshold is set to the maximum output activation to retrieve all words that phonetically best match the input word.

```
First letter of code is set to first letter of word
For all remaining word letters {
   if letter maps to 0 then skip
   if letter maps to same code as previous letter then skip
   Set next code character to value of letter mapping in table 2
}
If the code has less than 4 characters then pad with 0s
Truncate the code at 4 characters
```

Fig. 3. Figure listing our code generation algorithm in pseudocode. Skip jumps to the next loop iteration.

Integrating the Modules We produce three separate scores for the Hamming Distance, shifting n-gram and phonetic modules. We separate the Hamming Distance and n-gram scores so the system can utilise the best match and overcome the four forms of typing-error. We add the Soundex score to each producing two word scores. The overall word score is the maximum of these two values. We normalise all scores to ensure that none of the three modules biases the overall score. The score for the word is then given by (2).

$$\begin{aligned} \text{Score} = max((&\text{ScoreHamming} + \text{ScorePhonetic}), \\ &(\text{ScoreN-gram} + \text{ScorePhonetic})) \end{aligned} \tag{2}$$

2 Evaluation - Quality of Retrieval

We extracted 583 spelling errors from the Aspell [1] and Damerau [3] word lists and combined 17 spelling errors we extracted from the MS word 2000 autocorrect list to give 600 spelling errors. We counted the number of times each algorithm suggested the exact correct spelling among the top 10 matches and also the number of times the exact correct spelling was placed first. We include the score for MS Word 97, MS Word 2000 and Aspell spell-checkers for a benchmark comparison. We used the standard supplied dictionaries for both MS Word and Aspell. For all other evaluated methodologies we used the standard UNIX dictionary augmented with the correct spelling for each of our misspellings giving 29,178 words in total. We note that the dictionary and spelling errors contain many morphemes (singular/plural nouns, verb tenses etc) and we only counted the exact match. The spelling errors range from 1 to 5 error combinations.

3 Analysis

If we consider the final column of table 3, we can see that our hybrid implementation has the joint highest recall with Aspell. Our stated aim in the Introduction was high recall accuracy. Both Aspell and MS Word 2000 have more first place matches than our method. However, MS Word is optimised for first

Method	Found (Top 10)	Found (Position 1)	Present	Not Found	% Recall (Top 10)
Hybrid	558	368	6	36	93.9
Aspell	558	429	6	36	93.9
Word 2k	510	432	17	73	87.5
Word 97	504	415	15	81	86.1
Edit	510	367	6	84	85.6
Agrep	481	303	6	113	80.1

Table 3. The table indicates the recall accuracy of the methodologies evaluated. The present column indicates the number of misspellings present in the dictionary of each method, e.g, 'imbed' included in the dictionary along with 'embed'.

place retrieval and Aspell relies on a much larger rule base than we use. We have minimised our rule base for speed and yet achieved equivalent overall recall. The user will see the correct spelling in the top 10 matches an equal number of times for both Aspell and our method. We note that both Aspell and MS Word were using their standard dictionaries but we assume a valid comparison.

4 Conclusion

Spell checkers are somewhat dependent on the words in the lexicon. Some words have very few words spelt similarly, so even multiple mistakes will retrieve the correct word. Other words will have many similarly spelled words so one error may make correction difficult or impossible. Of the techniques evaluated, our hybrid approach had the joint highest recall rate at 93.9%. Humans averaged 74% for isolated word-spelling correction [4]. Kukich [4] posits that ideal isolated word-error correctors should exceed 90% when multiple matches are returned.

References

1. Aspell. Web page http://aspell.sourceforge.net/.
2. J. Austin. Distributed associative memories for high speed symbolic reasoning. In R. Sun and F. Alexandre, editors, *IJCAI '95 Working Notes of Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, pages 87–93, Montreal, Quebec, Aug. 1995.
3. F. Damerau. A technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176, 1964.
4. K. Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377–439, 1992.
5. J. R. Ullman. A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words. *Computer Journal*, 20(2):141–147, may 1977.
6. S. Wu and U. Manber. AGREP - A Fast Approximate Pattern Matching Tool. In *Usenix Winter 1992 Technical Conference*, pages 153–162, San Francisco, CA, Jan. 1992.
7. S. Wu and U. Manber. Fast Text Searching With Errors. *Communications of the ACM*, 35, Oct. 1992.