Text Induced Spelling Correction

Martin REYNAERT
Induction of Linguistic Knowledge, Computational Linguistics and AI, Tilburg University
Warandelaan 2, 5000 LE Tilburg,
The Netherlands,
reynaert@uvt.nl

Abstract

We present TISC, a language-independent and context-sensitive spelling checking and correction system designed to facilitate the automatic removal of non-word spelling errors in large corpora. Its lexicon is derived from a very large corpus of raw text, without supervision, and contains word unigrams and word bigrams. It is stored in a novel representation based on a purpose-built hashing function, which provides a fast and computationally tractable way of checking whether a particular word form likely constitutes a spelling error and of retrieving correction candidates. The system employs input context and lexicon evidence to automatically propose a limited number of ranked correction candidates when insufficient information for an unambiguous decision on a single correction is available. We describe the implemented prototype and evaluate it on English and Dutch text, containing real-world errors in more or less limited contexts. The results are compared with those of the isolated word spelling checking programs ISPELL and the MICROSOFT PROOFING TOOLS (MPT).

1 Introduction

All large text corpora contain spelling errors. People's internal language models, whatever the form they may take, are so robust that these errors do not necessarily affect them, nor are errors normally passed on and incorporated in the next generation's language models. Language technologists today derive statistical language models from corpora, accumulations of texts produced by whatever subset of a language community in electronic form. Errors present in these corpora are therefore passed on to the 'next generation' and, given the nature of electronic text, copying or archiving does not rectify these errors and transmission noise can only aggravate them. These errors accumulate and must influence to a certain degree our statistical language models or information-theoretic measures. They enhance the language sparseness problem in that they may take away valuable information, steal as it were, from the statistical weight of the correct form. In n-gram models the relative effect of this is further enhanced.

We think state-of-the-art spelling checking and correction systems are not equal to the task of cleaning up large corpora in that they are focused on achieving high recall to the detriment of precision. This means that for every error potentially corrected, a higher number would be erroneously replaced, if they did allow for automatic replacement. Automatic correction would certainly require a level of precision where more errors are removed than correct words replaced. We claim the algorithm presented here is a step in that direction. We further claim that the algorithm is to a large degree language-independent. To substantiate this we present and evaluate an English as well as a Dutch version, which differ only in the corpus-derived information available to them.

Section 2 presents our Text Induced Spelling Correction algorithm (TISC), while the corpus-derived components are discussed in section 3. We outline the implementation in section 4 and devote section 5 to the evaluation of both language versions and the discussion of the results.

2 The correction algorithm

We develop the idea of using the corpus itself as the basis on which to build a spelling correction system, since, after all, for every erroneously spelled form, the corpus should contain far more counterexamples of the adequate form.

2.1 Anagram hashing

We hit upon the idea of trying to line up all those forms present in the corpus that consist of the same set of characters so as to use that as the basis for a corpus-derived lexicon. A means to do this in a completely unsupervised way was found in the theory of hashing, be it in the 'bad' part of it, in the normally avoided generation of collisions. Collisions occur when the mathematical function used to bin the information, puts more than one item of information in a single bin (Knuth, 1981). The mathematically simple function introduced and exploited here does precisely that, for all strings containing the precise same set of characters.

So, for each word type or word type combination (compound or word bigram) to be included in the TISC lexicon, we obtain a numeric value, which will serve as the hash key. The formula represents the mathematical function we devised to do this, where f is a particular numerical value assigned to each character in the alphabet and c_1 to $c_{|w|}$ the actual characters in the input string w.

$$Key(w) = \sum_{i=1}^{|w|} f(c_i)^n$$

In practice, we use the ISO Latin-1 code value of each character in the string raised to a power n. We currently use 5 as the value for n. This was empirically derived, lower values do not produce collisions between anagrams only. The rather large natural number produced by this function in effect inflates the difference between any two characters to such a degree, that all strings containing the same set of characters receive the same natural number. This means that all anagrams, words consisting of a particular set of characters and present in the lexicon, will be identified through their common numeric value. So, in that the collisions produced by this function identify anagrams, we refer to this as an *anagram hash* and to the numeric values obtained as the *anagram keys*.

In the implementation we used chaining for collision resolution, as the anagram keys and their associated word forms are there stored in a regular hash. The anagram key will enable us to look up immediately whether any string consisting of the same character set as the input string was encountered in the corpus. When not present in the lexicon, close (numeric) neighbours might very well be present, and simple arithmetic will allow us to identify and retrieve these.

It is this novel representation that makes the implementation computationally tractable. The net effect of obtaining anagram hash key values is that it provides a cheap abstraction

Anagram key	anagrams
100036040884	routt, rutto, tortu, trout, tutor
108656935573	a trout, a tutor, art out, at tour, out art, rat out, rout at, tar out, tour at, trout a, tutor a
122746224841	rout the, the rout, the tour, tour the, tout her, trout he, true hot
124762035573	art unto, or taunt, taunt or, trout an, un trato
139280607949	or statue, our state, our taste, ouster at, out rates, out tears, rates out, routes at, sea trout, stare out, state our, statue or, taste our, tears out, touts are
139833841641	red trout
140937036472	ours that, out trash, sour that, that ours, that sour, trash out, trout has
163637465298	fried trout, if tortured, tortured if, turf editor, tutor fired
166797995067	out resort, roster out, routers to, sore trout, store tour, to routers, to trouser, trout rose, true roots

Table 1: Extract from a TISC lexicon with the anagram keys and associated, chained anagrams

from the surface sequence of characters which furthermore allows, through simple addition, subtraction or both, for moving from one particular combination of characters to another. The numeric distance obtained by means of this abstraction will in all cases be exactly the same for e.g. the difference between 'randomise' and 'randomize' or any other verb possibly ending in '-ise' or '-ize'. The same goes for all systematic differences between e.g. American and British English (think of single or double 'l' or 'ou' versus 'o').

Anagram key based spelling correction is an inexpensive solution to the string correction problem as it does not entail expensive searching: it uses the non-search strategy implied in hashing.

2.2 Anagram key based correction

Based on a word form's anagram key it becomes possible to systematically query the lexicon for any variants present, be they morphological, typographical or orthographical. These variants can all be seen as variations of the usual taxonomy in terms of *trasnpositions, *deletons, *inserrtions or *substatutions (Damerau, 1964).

transpositions These we get for free: they have the same anagram key value, so when queried, the lexicon returns the correct form and its anagrams (if any).

deletions We iterate over the alphabet and query the lexicon for the input word anagram value plus each value from the alphabet.

insertions We iterate over the list of anagram values for the character unigrams and bigrams collected from the input type and query the lexicon for the input word anagram value minus each of these values.

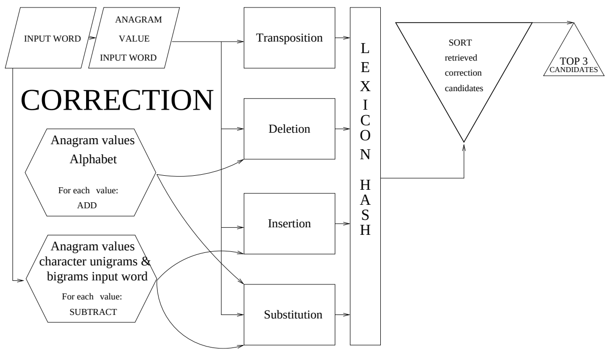

Figure 1: The correction module

substitutions We iterate over both lists adding each value from the one and subtracting each value of the second to the input word anagram value and repeatedly query the lexicon.

We thus retrieve from the lexicon all numerical near-neighbours (NNNs) and apply standard string matching techniques to store those that either in front or back match the input type for a specific amount of characters, depending on the input type's length. After doing so, we iterate over the list of NNNs obtained and upgrade the actual retrieval counts for those that have the greater substring matches and whose Levenshtein distance (LD) (Levenshtein, 1965) does not exceed 4, as the algorithm is not in itself limited to a particular LD. The elements of this list have thereby been ranked and the top n are then proposed as correction candidates. This ranking is an automatic side effect of the algorithm which produces more hits on the actual nearest NNN's. A deletion error, e.g. such as **cateory*, will return the correct 'category' on the basis of adding the value for 'g' as well as of substituting the value for 'e' with that for 'eg' and substituting the value for 'o' with that for 'go'. The redundancy inherent to our algorithm thereby produces the desirable side-effect of converging on what is usually the best correction candidate, besides the actual input string itself, if present in the lexicon.

3 TISC corpus-derived components

3.1 The Lexicon

The English corpus we used was the New York Times (1994-2002) material available in the LDC Gigaword Corpus (NYT) (Graff, 2003). For Dutch we used both the ILK Corpus[1] and the Twente Corpus[2] (TWC). Statistics on these corpora are presented in table 2.

Corpus	NYT	ILK-TWC
language	English	Dutch
tokens	1,106,376,695	681,686,340
bigrams (frq.>2)	11,246,986	9,927,378
derived unigrams	672,502	861,604
keys/anagrams	10,287,826	9,000,131

Table 2: Statistics of NYT and ILK-TWENTE corpus and lexicon

A TISC lexicon is derived from a large corpus of tokenised, but otherwise raw text, from which all XML or other tags have been discarded. We normalise the corpus by replacing all word-external punctuation by a single unique mark, as well as all digits and numbers by another. We apply a rule-based tokenizer and use the CMU Statistical Toolkit for deriving a bigram frequency list from the corpus (Clarkson and Rosenfeld, 1997). We discard the tail of the bigram list below a threshold frequency. Apart from the fact that this reduces the list to a size manageable for a standard PC with 1 gigabyte of RAM, it also ensures we do not incorporate the bulk of erroneous types present in the corpus. Next the frequency information is discarded and a unigram list derived from the retained part of the bigram list. We lowercase the unigram list and concatenate the three lists obtained, removing any doubles. We finally compute the anagram key values for the unigram/bigram list. Together, the anagram keys and their lined-up unigrams or bigrams (no unigrams can line up with bigrams as the space is regarded as a character in its own right) constitute the lexicon. Note that the lexicon will contain names and higher frequency errors.

3.2 The alphabet

Transformations on the word type to be evaluated are necessary in order to identify correction candidates. These transformations occur on the anagram key of the word type under consideration on the basis of numeric values for the alphabet used, which are read in at the start of run time. Our alphabet consists of the anagram key values for all character unigrams and character bigrams we want to work with. These have been derived from character unigram and bigram counts on the corpus.

[1]`http://ilk.uvt.nl/ilkcorpus/`

[2]`http://wwwhome.cs.utwente.nl/~druid/TwNC/TwNC-main.html`

3.3 The cooccurrence information

From the word bigram and unigram lists we derive cooccurrence information for all the word types present. For each word type we count the number of times it forms the:

- left part of a compound (LPC)
- right part of a compound (RPC)
- left part of a bigram (LPB)
- right part of a bigram (RPB)

Note that these cooccurrence counts (COOC) are counts on word types and not on word tokens. The COOC table contains only the counts per word-type, not the actual cooccurring word types.

4 TISC: the implementation

4.1 Zipf filters

Recall that Zipf stated that the frequency of a word is inversely proportional to its length (Zipf, 1935). This implies that we should expect to see more combinations of any given short word, be it in bigrams or as part of compounds, than of longer words. A long compound, e.g. one composed of three or more shorter words, cannot reasonably be expected to combine with very many more words. Short words can be expected to combine in a myriad of ways, be it as part of compounds or of numerous bigrams. It is this idea we exploit in what we would like to call the *Zipf Filters* implemented in our prototype. We make the number of expected cooccurrences of a word dependent on the length of the word form. This then allows to detect anomalies in the COOCs for particular word types. We posit a particular amount of times a string or substring is seen as sufficient to conclude the string is likely well-formed as it is highly productive. To this end we take a constant, which is higher for the shorter strings and lower beyond a particular amount of characters, divided by the number of characters in the string, or the string's length. We compare the COOCs of a string to be evaluated with the outcome of this calculation and accept the string as being well-formed when the COOCs are higher, reject and thus send on to the correction module, when lower.

4.2 Compound splitting

Given that a language such as Dutch to a large degree allows for compounding, any text may contain quite a number of previously unseen compounds. While iterating over the input word string to compute its anagram value, TISC repeatedly queries the lexicon to check for the presence of the substring handled so far. If this is successful for the whole string, the substrings, if any, which show the best balance between length and COOCs are stored with their anagram values. If no full parse was possible, the process is repeated from right to left and a decision made over both the left-right and right-left parses and the split deemed most usable stored. TISC proposes a single particular split to be further provided to the checking and correction modules. The implementation currently allows for only a split in a left and right part.

4.3 Checking

The input text is first fully analysed: anagram values are added to the type list, frequencies of types and their compounding parts tallied, track kept of how many times the type was capitalised, recurrent LPC's not in the lexicon stored. Then, all the types are sent to the spelling checking module. Since we cannot content ourselves with simply checking whether a type is present in the dictionary or not, we query the cooccurrence information table to see whether the particular type's COOCs conform to our expectation of how many times a type of the given length should have been incorporated in the lexicon, i.e. the expectancy level or threshold set by the Zipf filter. If this is the case, the type is not further evaluated, which we will refer to as 'let go'. If not, the COOCs for its LPC and the RPC are evaluated against the threshold. We do not, at this stage, want to risk to lose too many of the erroneous types, so the level of expectancy is set rather high. We simultaneously check whether perhaps the lexicon contains possible bigrams based on the type's anagram key value with the value for a space added. All the types which did not conform to the expected levels or were found to be present with an additional space, are further evaluated. Further checks are:

- extra-space cases: If it turns out the lexicon contains only the inverted form with the added space (e.g. 'koffiebekertje' [coffee cup]: not in the lexicon, but 'bekertje koffie' [cup of coffee] is present), we accept the form as being correct, the rest are further evaluated.
- whether perhaps the LPC was seen in various other input text compounds or whether the RPC was perhaps seen as a word in its

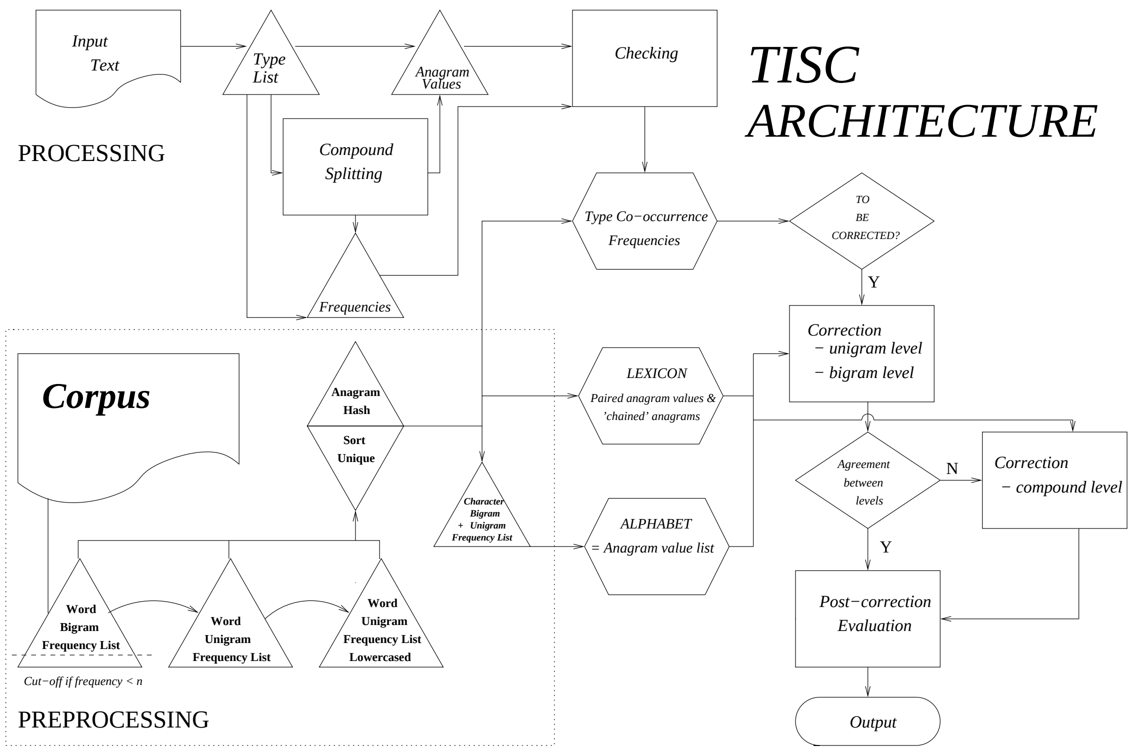
Input Text
Type List
Anagram Values
Checking
TISC ARCHITECTURE
PROCESSING
Compound Splitting
Type Co−occurrence Frequencies
TO BE CORRECTED?
Y
Frequencies
Correction − unigram level − bigram level
Corpus
Anagram Hash
Sort Unique
LEXICON
Paired anagram values & 'chained' anagrams
Agreement between levels
N
Correction − compound level
Character Bigram + Unigram Frequency List
ALPHABET = Anagram value list
Y
Word Bigram Frequency List
Word Unigram Frequency List
Word Unigram Frequency List Lowercased
Post−correction Evaluation
Cut−off if frequency < n
PREPROCESSING
Output

Figure 2: The TISC architecture

own right with a given frequency in the input text, the other part's COOCs conforming. Again those passing this test are let go.

- whether perhaps the COOCs for the LPC with first or all characters upper-cased conform to expectance.
- if the input type contains a dash, we check whether the COOCs for the type without the dash conform. Or perhaps whether the type without the dash but with an extra space is present in the lexicon.
- finally we check those forms for which the cooccurrence table contains no information at all. If the COOCs for their LPC and RPC exceed a high expectancy threshold, these are let go too.

All types not let go by one of these checks are sent on to the correction module.

4.4 Correction

By default, TISC's correction works on two levels, a third being invoked when these do not return satisfactory results. The unigram level consists of two tiers: unigram correction on the basis of the lexicon and on the basis of the list of input context derived types and compounding parts (with frequency threshold). On the bigram level, TISC performs context-dependent error correction, to some extent. It examines the 4 bigrams contained within a 2-1-2 window around the type in the input text (e.g. the green **bottel* was empty → the **bottel*, green **bottel*, **bottel* was, **bottel* empty). The only difference with the unigram correction module lies in the fact that for the 4 bigrams sent through the correction loop, all the correction candidates retrieved are stored in the same list. This produces more reliable counts after upgrading. After correction on these levels, the output candidates are compared and if both levels concur, i.e. the same candidate(s) were returned, they are accepted if they differ from the input type, or rejected (and 'let go') if not. When no output is returned by the unigram and bigram correction levels, or the results of these do not concur, the type is further checked on the third level, that of its substrings, i.e. the compounding parts returned by the compound splitter. The compound correction level treats both LPC and RPC as words in their own right, queries the system for correction candidates in the same way as on the unigram level for both parts and finally concatenates the top candidates returned and proposes these as correction candidates. Given a sufficiently high frequency in the input text of the correct form for an incorrect compounding part, this may enable the

	Metro1	Metro2	Reut1	Reut2
key code	M1	M2	3P	1P
context	article	article	3 par.	1 par.
tokens	21,919	25,750	97,432	40,349
types	5,747	6,441	15,341	9,590
errors	129	123	1,222	1,222
error/type	2.25%	1.9%	8%	12.7%

Table 3: Statistics of Metro and Reuters evaluation files

system to correct the error even if the correct form is not present in the lexicon.

5 Evaluation

5.1 Test settings

For TISC we report learning curves obtained by varying the threshold at which the corpora's bigram lists were truncated (Frequencies: 3, 5, 10, 20, 30, 40, 50 and 100 for NYT, 3–10 and 15 for ILK-TWENTE). The implementation used was the same for both languages as it contains no provisions specific to either. Both ISPELL and MPT were run with their standard US and standard Dutch dictionaries, the first in batch mode, the second manually emulating ISPELL's output for automatic evaluation purposes.

5.2 Composition of the evaluation files

For evaluation purposes, we proofread the Dutch version of the newspaper *Metro* during April-May 2003 and collected the non-word errors encountered. This amounts to 129 non-word errors, which were extracted from the on-line version with the full article they appeared in. We used this first batch (Metro1) for development purposes. A second bout of proofreading yielded a second, similar batch, which we reserved for testing purposes only (Metro2). We report the scores on both batches.

For evaluating the English version of TISC we manually collected 1093 erroneous types from the alphabetically sorted unigram frequency list of the Reuters Corpus (Lewis et al., 2003). We took care not to collect only very low frequency items. We then extracted their contexts from the tokenized corpus. The context ran to the paragraph containing the error, as well as the paragraphs preceding and following it, in all, almost 100.000 words of running text. This we proofread, which yielded another 105 errors, bringing the count up to 1198. A preliminary Ispell run finally yielded another 24 errors we had overlooked. We ran our evaluation test

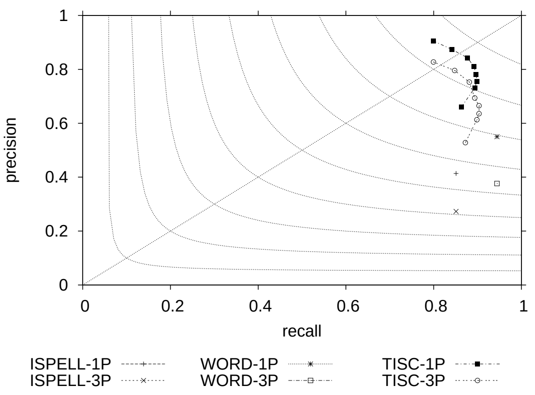

Figure 3: Evaluation results: English

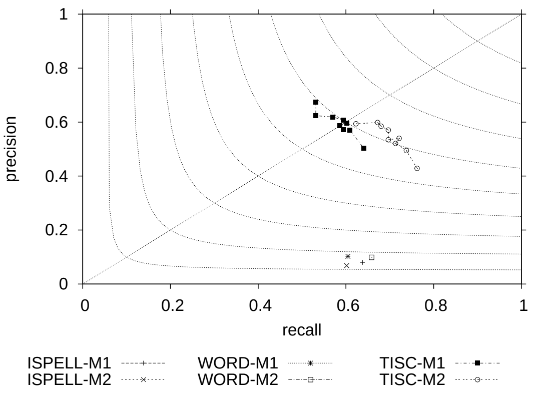

Figure 4: Evaluation results: Dutch

with these 1222 known errors. Evaluation results presented are those for the larger (Reut1) and a more limited context (Reut2), namely only those paragraphs actually containing errors. Statistics on the evaluation files are presented in table 3.

5.3 Scoring and evaluation results

We measure performance in terms of the F-score. Given that the three systems are presented with errors in a context, we do not solely measure their ability to correct incorrect forms, but also to discern between correct and incorrect input forms. The fact that the context is limited means that the distribution of errors versus non-errors here is seriously skewed compared to their actual distribution in the full corpora. Of the word forms for which correction candidates are returned, we check if the output contains the correct form. If so, the score for successful correction (recall) is augmented by one, no account being taken of the ranking of the correction candidates. For all the forms marked by ISPELL or MPT as 'not in the dictionary' the score for false positives (precision errors) is incremented by one. The same goes for

SYSTEM	I-3P	M-3P	T-3P	I-M2	M-M2	T-M2
thresh.	-	-	5	-	-	5
recall	0.85	**0.94**	0.85	0.60	0.66	**0.67**
precision	0.27	0.38	**0.80**	0.07	0.10	**0.60**
f-score	0.41	0.54	**0.82**	0.12	0.17	**0.63**
returned	3810	3065	**1302**	1087	820	**137**
corrected	1040	**1154**	1036	74	81	**82**
E.R.	2770	1911	**266**	1013	739	**55**
FAC	0.38	0.60	**3.89**	0.07	0.11	**1.49**

Table 4: Statistics of best test scores

those forms for which the three systems return correction candidates, but where the correct one is missing. Results presented were obtained on the types, for all three systems. Results for English are presented in figure 3, for Dutch: figure 4.

5.4 Discussion

For both languages, TISC's lower thresholded lexicons consistently produce the highest precision. Recall rises as the threshold is set higher, to drop again, as does precision, with more and more information not being available. As figure 3 shows, recall for ISPELL and MPT is unaffected by more context. TISC shows a slight gain on recall due to its context-awareness. Only the 3-paragraph context for **noe-emission* contains the correct form *no-emission*. The lexicon does not have this, but has *zero-emission*, on which both unigram and bigram levels concur. The input context evidence, which concurs with the compound level, allows TISC to nevertheless propose the correct form. More context does explain the drop in precision (TISC: 7%, ISPELL: 14%, MPT: 17%). More words to be checked create more opportunity to report false positives. This is also clearly demonstrated by the Dutch results, where the evaluation files contain an even lower error to type ratio. The drop in precision given more context seems to us to be the main cause of current spelling checking systems not being able to attain automatic correction levels of performance. If we accept the ratio of errors corrected versus correct words erroneously replaced (E.R.) to be a measure for a system's fitness for the purposes of automatic correction (FAC) (in the hypothetical case where the systems actually make the replacements and correction candidates are perfectly ranked), then TISC is fit (FAC > 1), for both languages, ISPELL and MPT are not.

6 Conclusion

We have presented TISC, a new algorithm for spelling checking and correction. We have outlined how the system is built up from scratch from a large corpus of raw text. We have introduced a novel representation for lexical information which allows for an exact calculation of the difference between two character strings. Not only does this make the problem computationally tractable, it also allows for building a fully scaled system. We have shown that incorporating word bigrams, cooccurrence information about individual word types and context information about the text to be spelling checked, all combine to enable automatic spelling correction. We have compared TISC with two state-of-the-art systems and shown that it consistently outperforms both, with balanced precision and recall, for both Dutch and English and for all evaluation sets.

Acknowledgements

Heartfelt thanks to my supervisors Prof. Dr. Walter Daelemans and Dr. Antal van den Bosch for their trust and support. This work was funded by the Netherlands Organisation for Scientific Research (NWO/FWO VNC 205-41-119).

References

P.R. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings ESCA Eurospeech 1997*.

Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, Volume 7, Issue 3 (March 1964):171 – 176.

David Graff. 2003. The New York Times Newswire Service. *English Gigaword LDC-2003T05*.

Donald E. Knuth, 1981. *Sorting and Searching*, volume 2 of *The Art of Computer Programming*, section 6.4, pages 513–558. Addison-Wesley, Reading, Massachusetts, second edition.

V.I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. In *Cybernetics and Control Theory*, volume 10(8), pages 707–710. Original in: Doklady Nauk SSSR 163(4): 845–848 (1965).

D. Lewis, Y.Yang, T.G. Rose, and F. Li. 2003. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*.

George Kingsley Zipf. 1935. *The psycho-biology of language: an introduction to dynamic philology*. The M.I.T. Press, Cambridge, MA, 1965 - 2nd. edition.