A Framework for Evaluating Text Correction

Robert Dale and George Narroway

Centre for Language Technology
Macquarie University
Sydney, Australia
Robert.Dale@mq.edu.au,George.Narroway@students.mq.edu.au

Abstract

Computer-based aids for writing assistance have been around since at least the early 1980s, focussing primarily on aspects such as spelling, grammar and style. The potential audience for such tools is very large indeed, and this is a clear case where we might expect to see language processing applications having a significant real-world impact. However, existing comparative evaluations of applications in this space are often no more than impressionistic and anecdotal reviews of commercial offerings as found in software magazines, making it hard to determine which approaches are superior. More rigorous evaluation in the scholarly literature has been held back in particular by the absence of shared datasets of texts marked-up with errors, and the lack of an agreed evaluation framework. Significant collections of publicly available data are now appearing; this paper describes a complementary evaluation framework, which has been piloted in the Helping Our Own shared task. The approach, which uses stand-off annotations for representing edits to text, can be used in a wide variety of text-correction tasks, and easily accommodates different error tagsets.

Keywords: Evaluation frameworks, error correction, replicability

1. Introduction

Computer-based aids for detecting and correcting errors in text have been around almost as long as the most primitive text processing applications. Spell checking techniques have been continually developed and improved since the 1960s; see (Kukich, 1992) for a still-pertinent review of techniques up to the late 1980s, and (Dale, to appear) for a survey of more recent work. The the set of tools known as the Unix Writer's Workbench (WWB; (Macdonald et al., 1982)) represented a significant step forward in the early 1980s, providing a collection of programs which used relatively simple pattern-matching techniques to detect a broad range of errors and infelicities in text; these quickly inspired a number of commercial applications that met the demand for proofing tools on the newly-appearing PC and Mac desktops. The technology took another leap later in the 1980s with IBM's development of the EPISTLE and CRITIQUE grammar checkers (Heidorn et al., 1982), the first broad coverage tools that used what we would today recognize as real parsing; these subsequently led to the grammar checker in Microsoft Word (Heidorn, 2000), probably the NLP application with the largest number of installed copies in the world.

The last two decades have seen a range of small-scale explorations of innovative grammar-checking techniques, but as far as broad-coverage grammar checking is concerned, progress appears to have stalled. This might be because the market dominance of the Microsoft product has killed any incentive that competitors might have to produce something better;[1] or it might be because further advances in the state of the art are just too difficult to achieve using the 'old school' rule-based techniques that underlie most approaches. But one other possible contributing factor is the difficulty of demonstrating that a new solution does indeed perform better than what has gone before. To do this requires both a shared dataset of spelling, grammar or style errors and their corrections, and an agreed framework for evaluating performance; unfortunately, both of these resources have been conspicuously absent.

But things are looking up. In particular, publicly-available datasets of texts marked-up with second-language learner errors have started to appear, a timely development given the significant interest in the last few years in building applications that aim to help this audience. In part motivated by these developments, this paper presents an attempt to provide a complementary resource in the form of a flexible evaluation framework that is tailored to the specific needs of the task of text correction, while allowing the use of a wide range of error annotation schemas. The code and various sample data sets are freely available.[2]

2. Desiderata

The framework presented here was developed in the context of the HOO (Helping Our Own) shared task, whose pilot run was reported on as part of the 2011 Generation Challenges at the European Natural Language Generation Workshop (see (Dale and Kilgarriff, 2011). The stated aim of HOO is to encourage the development of technologies that help members of the NLP community write better papers about NLP (hence 'helping our own'); we set up the initiative (see (Dale and Kilgarriff, 2010)) in the hope that focussing on this particular user community would result in greater 'buy-in' by both developers and potential users. Our vision encompasses a very broad range of assistive technologies to help with aspects of the writing task at many levels; but for the initial rounds of the shared task, we chose to focus on what we might consider to be 'tactical' aspects of writing, conventionally referred to as spelling, grammar and style.

For the pilot run of the HOO shared task, we used texts that had been produced by non-native speakers of English;

[1] See (Dale, 2004) for an analysis along these lines.

[2] See www.correcttext.org.

```
<edit type="IJ" index="0001-0006"
      start="771" end="782">
   <original>electronics</original>
   <corrections>
      <correction>electronic</correction>
   </corrections>
</edit>
<edit type="RP" index="0001-0007"
      start="1387" end="1388">
   <original>;</original>
   <corrections>
      <correction>.</correction>
   </corrections>
</edit>
<edit type="MY" index="0001-0004"
      start="631" end="631">
   <original><empty/></original>
   <corrections>
      <correction/>
      <correction>both </correction>
   </corrections>
</edit>
<edit type="RV" index="0001-0005"
      start="713" end="718">
   <original>carry</original>
   <corrections>
      <correction/>
      <correction>contain</correction>
   </corrections>
</edit>
```

Figure 1: Some gold-standard edit structures.

these were marked up using an error tagset that had been designed with this kind of text in mind. However, from the outset we have been mindful that different text correction tasks might warrant different tagsets at varying granularities, so our annotation scheme has been deliberately designed to offer flexibility on this front; this is described in Section 3.

Our inclusion of stylistic aspects of text correction immediately brings to the fore some difficult questions for evaluation. It might be argued that, where spelling and grammar are concerned, there is indeed a correct answer, and so a system can be evaluated on both whether it detects the incorrect usage and whether it provides the stipulated correction. Stylistic questions, however, are much less clear cut;[3] consequently, our evaluation framework allows for both multiple possible corrections and for optional corrections for a given error or infelicity. Given these phenomena, it may be more appropriate to think of the general task we are interested in as being more generally about 'text massaging' rather than 'text correction', and more appropriate to use the term 'infelicity' rather than the term 'error'. Our evaluation metrics are described in Section 4.

3. Annotating Infelicities

To use our framework, texts are marked up using inline annotations that indicate the extent of an identified infelicity in the text, along with the assignation of a type, and zero or more possible corrections.[4] These inline annotations are then automatically extracted from the texts to produce a set of stand-off annotations that can be used for evaluation purposes. Figure 1 shows some example annotations.[5]

Each `<edit>` element has an `index` attribute that uniquely identifies the edit, a `type` attribute which indicates the type of the error found or correction made,[6] and a pair of offsets that specify the character positions in the source text file of the `start` and `end` of the character sequence that is affected by the edit. The set of types to be used is defined independently of the annotation schema, so that other tagsets can be used. The embedded `<original>` element contains the text span that is subject to correction. Each `<edit>` element usually also contains a `<corrections>` element, which lists one or more possible corrections for the problematic text span.

There are a number of complicating circumstances we have to deal with. First, there may be **multiple valid corrections**. This is not just a consequence of our desire to include classes of infelicitious usage where there is no single best correction. The requirement is already present in any attempt to handle grammatical number agreement issues, for example, where an instance of number disagreement might be repaired by making the affected items either singular or plural. Also, it is usually not possible to consider the list of corrections we provide as being exhaustive.

A correction may also be considered **optional**. In such cases we view the first listed correction as a null correction (in other words, one of the multiple possible corrections is to leave things as they are); the last two edits in Figure 1 provide examples. When an edit contains an optional correction, we call the edit an **optional edit**. if the edit contains no optional corrections, then it is a **mandatory edit**. Note that deletions and insertions, as well as replacements, may be optional.

Sometimes edits may be interdependent: making one change requires that another also be made. Edits which are connected together in this way are indicated via indexed `cset` attributes (for **consistency set**). The most obvious case of this is where there is requirement for consistency in the use of some form (for example, the hyphenation of a term) across a document; each such instance will then belong to the same `cset` (and consequently there can be many members in a `cset`). Another situation that can be handled using `cset`s is that of grammatical number agreement. In

[3] In fact, questions of spelling and grammar are not always so 'black and white' either, of course.

[4] We allow the possibility of zero corrections to cater for circumstances where the annotator has determined that something is wrong, but is not able to determine what an appropriate correction would be.

[5] The types here are: IJ = incorrect adjective inflection, RP = replace punctuation, MY = missing adverb, and RV = replace verb.

[6] The set of types used here is borrowed, with some very minor changes, from the Cambridge University Press Error Coding System described in (Nicholls, 2003), and used with permission of Cambridge University Press.

such a case, there are usually two possible corrections, but the items affected may be separated in the text, requiring two separate edits to be made; these are connected in the annotations by a `cset`.

Finally, there are cases where our annotators have determined that something is wrong, but are not able to determine what the correction should be. There are two common circumstances where this occurs: (1) a word or fragment of text is missing, but it is not clear what the missing text should be; or (2) a fragment of text contains a complex error, but it is not obvious how to repair the error. These two cases are represented by omitting the `corrections` element.

All of these phenomena complicate the process of evaluation, which we turn to next.

4. Evaluation

The annotation schema used and the evaluation metrics to be applied are related but logically separate, allowing additional metrics to be added. In the current release of the evaluation software, the approach we take borrows much from the evaluation of named-entity recognition tasks. A number of characteristics of the present task mean that it is not straighrforward to evaluate performance:

1. Reasonable people may disagree as to whether an edit indicated in the gold standard is necessary.

2. Reasonable people may disagree as to whether the gold standard contains all the edits that should be made.

3. Even where there is an agreement that an edit should be made, reasonable people may disagree as to what the correction should be, and participating systems may offer corrections other than those provided in the gold standard.

4. A system's view of the extent corresponding to an edit may not agree with the extent indicated in the gold standard.

In what follows, we approach evaluation much as if it was a named-entity-mention recognition task, where the markables are the named-entity mentions to be detected in the source text. This allows us to use fairly standard approaches to measuring success in recognizing the presence of markables in the source.

There are a number of aspects of system performance for which we can derive scores. **Detection** measures a system's ability to determine that an edit is required at some point in the text; **Recognition** measures whether the extent of the source text that requires editing is identified correctly; and **Correction** measures a system's ability to offer a correction that is amongst the corrections provided in the gold standard. Detection is effectively 'lenient recognition', allowing for the possibility that the system and the gold standard may not agree on the precise extent of a correction. Systems can be scored on a text-by-text basis, on a data set as a whole, or on the basis of individually specified error types.

For each pairing of gold standard data and system output,we compute two **alignment sets**, recording the correspondences between the two sets of edits. The **strict alignment set** contains those alignments whose extents match perfectly; the **lenient alignment set** contains those alignments that involve some overlap. **Unaligned edits** are edits which do not appear in the lenient alignment set: an unaligned system edit corresponds to a **spurious** edit, and an unaligned gold-standard edit corresponds to a **missing** edit. Note that missing edits are of two types, depending on whether the gold-standard edit corresponds to an optional edit or a mandatory edit. A system should not be penalised for failing to provide a correction for a markable where the gold standard considers the edit to be optional. To manage the impact of this on scoring, we keep track of the number of **missing optional edits**.

4.1. Detection

For a given pair of edit sets, a gold standard edit is considered **detected** if there is at least one alignment in the lenient alignment set that contains that edit. We calculate Precision as the proportion of edits found by the system that correspond to gold-standard edits,[7]

$$P = \frac{\textit{\# detected edits}}{\textit{\# spurious edits} + \textit{\# detected edits}} \tag{1}$$

and Recall is calculated as:

$$R = \frac{\textit{\# detected edits}}{\textit{\# gold edits}} \tag{2}$$

as the proportion of gold-standard edits that were detected. However, under this regime, if all the gold edits are optional and none are detected by the system, then the system's Precison and Recall will both be zero. This is arguably unfair, since doing nothing in the face of an optional edit is perfectly acceptable; so, to accommodate this, we also compute scores 'with bonus', where a system also receives reward for optional edits where it does nothing.

$$P = \frac{\textit{\# detected} + \textit{\# missing optional}}{\textit{\# spurious} + \textit{\# detected} + \textit{\# missing optional}} \tag{3}$$

$$R = \frac{\textit{\# detected} + \textit{\# missing optional}}{\textit{\# gold edits}} \tag{4}$$

This has a more obvious impact when we score on a text-by-text basis, since the chances of a system proposing no edits for a single text are greater than the chances of the system proposing no edits for all texts.

The **Detection** score is then the harmonic mean (F-score).

$$\textit{DetectionScore} = 2 \times \frac{\textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}} \tag{5}$$

[7] In all computations of Precision (P) and Recall (R) we take the result of dividing zero by zero to equal 1, but for the computation of F-scores we take the result of dividing zero by zero to be zero.

4.2. Recognition

We consider a gold-standard edit to be **recognized** if it appears in the strict alignment set. *RecognitionScore* is defined to be 0 if there are no recognized edits for a given document; otherwise, we compute Precision as the proportion of system edits which correspond to gold-standard edits, and Recall as the propertion of gold-standard edits that are recognized. The **Recognition** score is again the harmonic mean; as for **Detection**, we compute a 'with bonus' variant that gives credit for missed optional edits.

4.3. Correction

For any given gold-standard edit, there may be multiple possible corrections. A system edit is considered a **valid correction** if it is strictly aligned, and the correction string that it contains is identical to one of the corrections provided in the gold standard edit. *CorrectionScore* is defined to be 0 if there are no recognized edits for a given document; otherwise, we compute Precision as the proportion of system edits for which the provided correction is valid, and Recall as the proportion of gold standard edits for which a valid correction is provided. The correction score is, as before, the harmonic mean, and we once again compute 'with-bonus' variants.

4.4. Type-based Evaluation

The metrics just described can be applied to all the annotated errors in an individual file or in a collection of files; however, we recognize that in some circumstances only errors of a particular type may be of interest. The evaluation of specific types of corrections is facilitated by the use of a control file that provides a one-to-many mapping between **reportable types** and **annotated types**. The idea here is that the annotations provided in the gold standard may be a finer level of granularity than we require for evaluation purposes; for example, the CLC tagset provides a number of different tags for errors that are concerned with the use of determiners, but we may prefer to collapse these into a single type. This information is recorded in a configuration file, which can be specified as a command line argument to the evaluation tool. Here's a typical line from a configuration file:

```
aggregate PUNCT RP MP UP
```

This indicates that the listed CLC error tags RP (replace punctuation), MP (missing punctuation) and UP (unnecessary punctuation) should be reported as PUNCT errors; a configuration file which contains only this line will result in scores being reported only for punctuation errors, and all other errors being ignored. A configuration file may list multiple reportable types, thus allowing for different approaches to taxonomising the space of errors.

5. Conclusions and Future Work

The evaluation framework described here was used successfully in the HOO Pilot Shared Task, in which six teams took part; see (Dale and Kilgarriff, 2011). The pilot used a a small collection of corrected text fragments drawn from the ACL Anthology, and used with the kind permission of their authors. Recently, two large corpora of error-tagged second language learner data have been publicly released: the CLC FCE Dataset contains 1,244 exam scripts from the Cambridge ESOL First Certificate in English (FCE) examination,[8] and the NUS Corpus of Learner English (NUCLE) contains 1,400 essays written by university students at NUS on a wide range of topics.[9] These two corpora use different tagsets, but can be straightforwardly converted into the annotation schema described here. At the time of writing, the second HOO Shared Task, which focuses on correcting preposition and determiner usage in second language learner texts (drawn from the FCE dataset) is underway: the high proportion of ESL errors that fall into these categories has led to significant interest in addressing these problems, but so far much of this work remains hard to compare because of the use of different data sets and evaluation procedures. The framework provided here, in conjunction with the newly available data sets, provides an opportunity for robust and objective comparisons of current and new approaches.

6. References

R Dale and A Kilgarriff. 2010. Helping Our Own: Text massaging for computational linguistics as a new shared task. In *Proceedings of the 6th International Natural Language Generation Conference*, pages 261–266, 7th-9th July 2010.

R Dale and A Kilgarriff. 2011. Helping our own: The HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*, 28th–30th September 2011.

R Dale. 2004. Industry watch. *Natural Language Engineering*, 10:91–94.

R Dale. to appear. Automated writing assistance. In R Mitkov, editor, *Oxford Handbook of Natural Language Processing*. Oxford University Press, second edition.

G E Heidorn, K Jensen, L A Miller, R J Byrd, and M Chodorow. 1982. The EPISTLE text-critiquing system. *IBM Systems Journal*, 21:305–326.

G Heidorn. 2000. Intelligent writing assistance. In R Dale, H Moisl, and H Somers, editors, *Handbook of Natural Language Processing*, pages 181–207. Marcel Dekker.

K Kukich. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439.

N Macdonald, L Frase, P Gingrich, and S Keenan. 1982. The Writer's Workbench: Computer aids for text analysis. *IEEE Transactions on Communications*, 30(1):105–110.

D Nicholls. 2003. The Cambridge Learner Corpus—error coding and analysis for lexicography and ELT. In D Archer, P Rayson, A Wilson, and T McEnery, editors, *Proceedings of the Corpus Linguistics 2003 Conference*, pages 572–581, 29th March–2nd April 2001.

[8] `http://ilexir.co.uk/applications/clc-fce-dataset.`

[9] `http://nlp.comp.nus.edu.sg/corpora.`