The HOO Pilot Data Set: Notes on Release 2.0

Robert Dale and George Narroway

June 21, 2011

Abstract

This document describes the first Helping Our Own (HOO) data set, to be used in the Pilot Round of the HOO Shared Task taking place in the second half of 2011. The aims of the present document are to explain the nature of the data that is being made available; to draw attention to various design decisions that were made in collecting and annotating the data; and to describe the approach that is being taken to evaluation.

1 Introduction

The Helping Our Own (HOO) Shared Task aims to promote the development of automated tools and techniques that can assist authors in the writing task. The task focusses specifically on writing within the natural language processing community, on the grounds that content matter familiar to Shared Task participants will be more engaging than content matter from another discipline. In addition, the ACL Anthology provides us with a large and freely-available collection of material in the appropriate domain and genre that can be used, for example, for language modelling; obtaining similar material in other disciplines is more difficult and potentially costly. A broader discussion of the background to the HOO task can be found in (Dale and Kilgarriff, 2010).

In this first pilot round of the task, we focus on errors and infelicities introduced into text by non-native speakers (NNSs) of English. While there are few native speakers who would not also have something to gain from the kinds of technologies we would like to see developed, the generally higher density of errors in texts authored by NNSs makes annotation of this material much more cost efficient than the annotation of native-speaker text. The focus on English texts is for purely pragmatic reasons; obviously one could in principle pursue the goals discussed here for other languages too.

These notes describe the data we are providing for the first run of the HOO task, and outline a number of decisions that were made in collating and preparing the data and the associated evaluation machinery. We see this round of the task as being largely exploratory in nature. In particular, there are a number of unresolved questions in regard to annotation and evaluation; we have taken the view that the best way to answer these questions is to encourage as many teams as possible to work with the data we are providing, so that we can use first-hand experiences to collectively learn what works and what doesn't.

Section 2 describes the overall data set that we provide; Section 3 discusses the approach taken to annotation, and the content of the annotations; Section 4 summarises our approach to evaluation; and Section 5 provides some concluding remarks.

The present document also contains some additional material in appendices. Appendices A and B list the Cambridge Learner Corpus (CLC) error tags, which are used as the basis of the scheme adopted here;[1] Appendix C provides some statistics on the initial data set; Appendix D describes how to use the evaluation tools we provide; and Appendix F lists known problems with the current framework and data set.

[1]The Cambridge University Press Error Coding System is copyright to Cambridge University Press and may only be used with their written permission. The coding is used to annotate the Cambridge Learner Corpus, which informs English Language Teaching materials published by Cambridge University Press.

2 The Data Set

The initial HOO data set consists of 19 files, each of approximately 1000 words in length. We refer to the content of each file as a **fragment**, since it contains the text corresponding to a fragment of a larger document. In each case, the **source document** for the fragment is a paper that has been published in the proceedings of a conference or a workshop of the Association for Computational Linguistics; the authors of these papers have kindly permitted their material to be used in the Shared Task.

Each fragment was annotated with a number of **edits** to correct errors and infelicities, as discussed further below, by two professional copy-editors. Collectively, the fragments contain a total of 1264 edits, or an average of 66 per file, with a minimum of 16 and a maximum of 100. This works out at an average of one edit every 15 words.

Each fragment is uniquely identifiable by a four-digit number used as the initial part of the names of the files that are associated with that fragment; the contents of a given file are identified by the format of the remainder of the file name. In what follows we use the fragment identifier '0001' as an example.

The complete text of each fragment is provided in two different versions, either of which may be used as input for processing by participating teams:

0001.txt: This contains the original text fragment as extracted from an authored document, in plain text form. An excerpt of such a fragment is shown in Figure 1.

0001S.txt: This contains a minimally-marked-up version of the original text fragment, in which sentence and paragraph breaks are marked using `<s>` and `<p>` tags respectively.[2] The entire text is also wrapped up within `<hoo>` tags. An excerpt of such a fragment is shown in Figure 2.[3] See Section 3.1 below for further comments on sentence segmentation.

These files contain one paragraph of text per line; line endings are Unix-style, i.e. a single linefeed character. The character encoding used is plain ASCII.[4]

Corresponding to each fragment, there is also a file containing, in stand-off markup format, the set of target edits for that file. For fragment 0001, the corresponding **gold-standard edits** file is **0001GE.xml**; an excerpt of this file is shown in Figure 3.[5] The output of participating systems will be compared against these files, whose contents we will refer to as **edit structures**.

Participating systems may choose to deliver their results in either one of two forms:

1. A set of plain text files that contain corrected text *in situ*; we provide a tool that extracts the changes made to produce a set of XML edit structures for evaluation.

2. A set of edit structures that encode the corrections their system makes.

There are advantages to providing the latter: in particular, edit structures provide a higher degree of fidelity in capturing the specific changes made, as discussed in Section 4.2. However, there may be a higher overhead incurred in constructing these, so some teams may prefer to provide plain corrected text.

Participating systems will then be evaluated by comparing the edits they make against the sets of gold-standard edits, as described in Section 4. The format of the gold-standard edits is described in detail in Section 3 below.

For reference only, we also provide for each fragment a gold-standard annotated version of the text built on top of the sentence-segmented fragment, with corrections marked in-line; for fragment 0001, the corresponding in-line annotated file is named **0001G.xml**. A CSS style sheet is provided that allows these versions of the data to be conveniently viewed in a web browser; Figure 4 shows an example, and Figure 5 shows what this looks like in an appropriately enabled browser.[6]

[2] Each `<s>` tag is paired with a `</s>` tag, and each `<p>` tag is paired with a `</p>` tag; we take this as a given in what follows, and refer to tag pairs simply via the name of the corresponding opening tag.

[3] The `<hoo>` tags are not shown here since this paragraph is neither at the beginning nor at the end of the document.

[4] Earlier versions of the released data contained some ISO-LATIN-1 characters. These have been replaced in the current release by their closest ASCII equivalents.

[5] The top-level element in each such file, not shown in this example, is `<edits>`.

[6] We are very grateful to Guy Lapalme at Université de Montréal for providing the various XSLT and CSS components to make this possible. Note that the pop-ups shown here work in Firefox but not in Internet Explorer; other browsers have not been tested.

```
Let us consider two cases where the pairs of multilingual inputs in English and Korean have
identical and different subjectivity meanings (Figure 1).  The first pair of texts carry a negative
sentiment about how the release of a new electronics device might affect an emerging business
market.  When a multilanguage-comparable system is inputted with such a pair, its output should
appropriately reflect the negative sentiment, and be identical for both texts.  The second pair
of texts share a similar positive sentiment about a mobile device's battery capacity but with
different strengths.  A good multilingual system must be able to identify the positive sentiments
and distinguish the differences in their intensities.
```

Figure 1: Part of an original text fragment.

```
<p><s>Let us consider two cases where the pairs of multilingual inputs in English and Korean have
identical and different subjectivity meanings (Figure 1).</s> <s>The first pair of texts carry
a negative sentiment about how the release of a new electronics device might affect an emerging
business market.</s> <s>When a multilanguage-comparable system is inputted with such a pair, its
output should appropriately reflect the negative sentiment, and be identical for both texts.</s>
<s>The second pair of texts share a similar positive sentiment about a mobile device's battery
capacity but with different strengths.</s> <s>A good multilingual system must be able to identify
the positive sentiments and distinguish the differences in their intensities.</s></p>
```

Figure 2: The sentence-segmented version of the text in Figure 1.

```
<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>
<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>
```

Figure 3: Some gold-standard edits for the text in Figure 1.

```
<p><s>Let us consider two cases where the pairs of multilingual inputs in English
and Korean have <edit type="MY"><empty/><corrections><correction/><correction>both
</correction></corrections></edit> identical and different subjectivity meanings (Figure 1).</s>
<s>The first pair of texts <edit type="RV">carry<corrections><correction/><correction>contain
</correction></corrections></edit> a negative sentiment about how the release of a new <edit
type="IJ">electronics<corrections><correction>electronic</correction></corrections></edit>
device might affect an emerging business market.</s> <s>When a multilanguage-comparable system
is inputted with such a pair, its output should appropriately reflect the negative sentiment, and
be identical for both texts.</s> <s>The second pair of texts share a similar positive sentiment
about a mobile device's battery capacity but with different strengths.</s> <s>A good multilingual
system must be able to identify the positive sentiments and distinguish the differences in their
intensities.</s></p>
```

Figure 4: The gold-standard version of the text in Figure 1 with inline annotations.

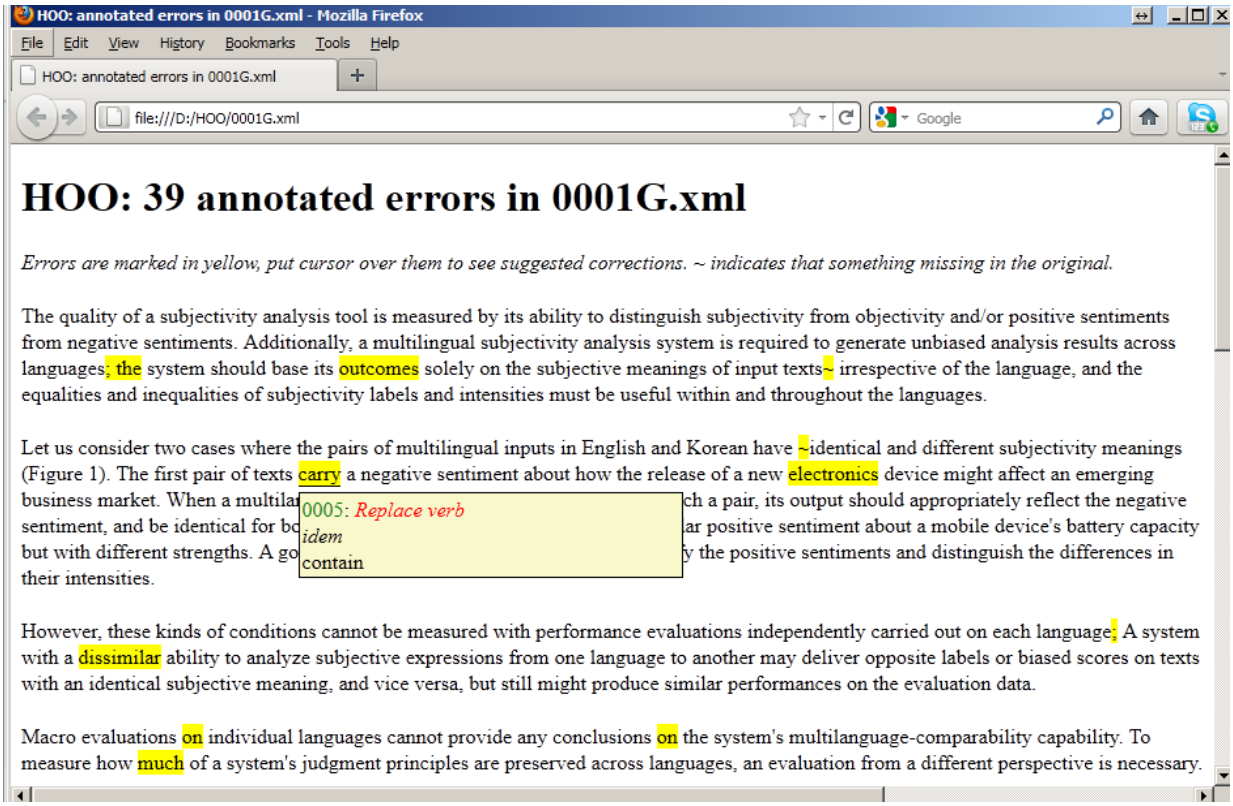

Figure 5: A browser view of the file in Figure 4.

3 Annotations

3.1 Sentence Segmentation

We expect that many participants will apply processing that works on a sentence-by-sentence basis, and so we have provided a version of the fragments which contain a gold-standard sentence segmentation. This segmentation was produced by first running a simple script to add <s> and </s> tags at likely places in the text, and then hand-correcting these annotations to deal with both (a) incorrectly inserted tags where a full stop has a function other than sentence termination, and (b) cases where it is fairly clear that the author intended a sentence break but failed to indicate this by the use of a full stop. It is not uncommon, for example, to find a comma being used in a location where all the other contextual evidence suggests that this should have been a full stop, as in the following example:

(1) `... said yesterday, Meanwhile ...`

Similarly, there are cases where the sentence-terminating puncuation mark has simply been omitted from the original text.

Of course, there are situations where our segmentation decisions can be called into question. In many cases, for example—and the instance above might be one such—it is plausible that what was really intended by the author was a semi-colon break. In the present example this would also require correcting the casing of *Meanwhile*, so here we are reasonably confident that the comma preceding *Meanwhile* really should have been a full stop, but it is hard to be 100% certain.

When sentence segmentation annotations are added to the data, we retain the inter-sentential spacing that was present in the original text. So, for the example above, the corresponding segmented text looks like the following:

(2) `... said yesterday,</s> <s>Meanwhile ...`

The point to note here is that the space between the closing </s> tag and the subsequent opening <s> tag is retained. This means that we can reconstruct the unsegmented original version of the text from the segmented version, and that character offsets across all versions of the fragment can be computed consistently.

Note that, in the current version of the data, headings and bullet list items are tagged as sentences.

3.2 The Annotation of Corrections

3.2.1 What Counts as a Markable?

We will refer to a span of source text for which an edit is proposed as a **markable**. By an **edit** we mean any change that is made to a text: from the outset, our intent has been to deal with textual modifications that go some way beyond the correction of, for example, grammatical errors. This decision presents us with a significant challenge. Whereas the presence of spelling and grammatical errors might seem to be something that competent speakers of a language would agree on, as soon as we go beyond such phenomena to encompass what we will sometimes refer to as 'stylistic infelicities', there is increasing scope for disagreement.[7]

As noted above, our data has been annotated by two professional copy-editors. They each annotated the texts for edits independently, and then they discussed disagreements between their annotations with the aim of reaching a consensus. In terms of deciding what should be marked for editing, the initially-proposed diagnostic was that the annotators should edit anything they felt corresponded to 'incorrect usage'. A brief perusal of the data will reveal that, not surprisingly, this is a very difficult notion to pin down precisely.

There are a reasonable number of cases where our annotators disagreed on whether an edit was necessary, and when they did agree that an edit should be made, there were also a reasonable number of cases where they proposed different corrections, even after discussion. Our annotation scheme thus allows for multiple possible corrections, and for optional corrections (i.e, where the text may also be left as is); clearly the presence of both these phenomena raises issues for evaluation, as discussed in Section 4 below.

[7]The notions of spelling error and grammatical error are not necessarily so clear cut either. For example, what counts as a spelling error in British English might be considered a correct spelling in US English. With respect to grammar, there can be reasonable disagreements as to whether, for example, the use of a particular preposition in some verb frame is a grammatical error or a collocational error.

3.2.2 The Extent of Markables

Conceptually, a correction to a text involves replacing a sequence of characters at some location with another sequence of characters. The **extent** of a piece of text to be edited is therefore defined by its start and end positions in the original text file; these are counted in integer values from the beginning of the file, starting at 0. Each printable character, including spaces and linefeeds, contributes to the calculation of offsets; however, any XML tags in the text do not contribute to these counts. Character positions can be thought of as index values that lie between pairs of characters. So, in the following text, the underlined extent has start position 0 and end position 3:

(3) `<s>The good life never ever ends.</s>`

As a shorthand, we write this extent as [0:3]. Because the replacement sequence of characters may not be the same length as the original sequence of characters, we always indicate extents by means of offsets into the original source file, not into a version of the file that has been changed to accommodate corrections. So, suppose in the fragment above the word *The* is replaced by the word *A*, and then the word *good* is replaced by the word *bad*; the extent that the second correction refers to is [4:8], since this is the span of text in the original file that it corrects.

In the limiting case of an insertion, a character sequence is inserted at an extent with zero length. Suppose we now want to change the text to read as follows:

(4) `<s>A very bad life never ever ends.</s>`

Then the extent of the insertion is [4:4]. Note that the insertion here includes a space, so that the resulting text maintains conventional interword separation.

There are some constraints on markables that we impose for a variety of technical reasons:

1. Markables may not cross sentence boundaries in the source text.
2. Markables may not overlap in their extents, which means in particular that two markables cannot begin at the same point, and a markable cannot be contained within another markable.

Generally, extents will consist of an integral number of tokens; so, for example, removing a hyphen from a word like *data-base* is not indicated by just deleting the hyphen, but by replacing the hyphenated word by its unhyphenated equivalent, *database*. Note, however, that the notion of tokenhood in play here is an intuitive one as exercised by the annotators; this may not alway agree with a given machine tokenisation algorithm, particularly where punctuation characters placed adjacent to words are concerned.

In general, only those tokens which are actually affected by a change—and not any surrounding context—are included in the markable extent. However, when there are multiple possible corrections at some point in a text, it is possible that one may require more tokens to change than another; in such a case, it is the sequence of source tokens affected by the longest correction that determines the markable extent, and each of the corrections listed will provide a replacement for the entire extent. Consider the following example:

(5)
Original Text:	. . . the ambiguity of the person name . . .
Correction #1:	. . . the ambiguity of the personal name . . .
Correction #2:	. . . the ambiguity of the person's name . . .
Correction #3:	. . . the ambiguity of personal names . . .

The first two corrections require only one word to be changed, but the third correction requires that three words in the original text be replaced by two words. Consequently, the edit structure for this correction takes the three words *the person name* as the corresponding markable for all three corrections:

(6)
```
<edit type="R" start="17" end="34">
   <original>the person name</original>
   <corrections>
      <correction>the personal name</correction>
      <correction>the person's name</correction>
      <correction>personal names</correction>
   </corrections>
</edit>
```

Note that each correction provides a replacement for all three words in the original text.

3.2.3 The Basic Format of Annotations

The general format of edits in the gold-standard edit files is exemplified by the following:

(7)
```
<edit index="0001-0006" type="IJ" start="771" end="782">
   <original>electronics</original>
   <corrections>
      <correction>electronic</correction>
   </corrections>
</edit>
```

To elaborate on each aspect of this representation:

1. The information concerning each markable is contained within an `<edit>` element.
2. Each `<edit>` element has an `index` attribute that uniquely identifies the edit; this index consists of a four-digit number corresponding to the identify of the fragment that the edit occurs in (here, `0001`), and a four-digit number corresponding to the ordinal position of this edit in the sequence of edits in that fragment (here, `0006`, indicating that it is the sixth edit in this text fragment). Index values will generally be omitted in subsequent examples in this document.
3. Each `<edit>` element has a `type` attribute that indicates the type of the error found or correction made. The full set of types used to annotate corrections is provided in Appendix A; in the present example, the tag `IJ` means Incorrect Adjectival Inflection.
4. Each `<edit>` element has 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 (here, 771 and 782 respectively). These offsets will generally be omitted in subsequent examples in this document.
5. Each `<edit>` element contains an embedded `<original>` element, which contains the text span that is subject to correction.
6. Each `<edit>` element may contain an embedded `<corrections>` element, which lists one or more possible corrections for the problematic text span that has been identified. Each possible correction in this list is surrounded by `<correction>` tags.

There are a number of respects in which an edit annotation may be more complex than the example above; we now discuss these in turn.

3.2.4 Handling Multiple Possible Corrections

As indicated earlier, we need to allow for the possibility that 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.

Example (8) provides a simple example of multiple corrections:[8]

(8)
```
<edit type="CC">
   <original>provides a relatively balanced corpus</original>
   <corrections>
      <correction>is a relatively balanced corpus,</correction>
      <correction>is relatively balanced,</correction>
   </corrections>
</edit>
```

[8]The `CC` error type used here means Compound Change; this is not an error type that exists in the CLC tagset we use, but is a tag we have added to deal with cases where the annotators felt either that a number of tags would be required to indicate the nature of the problem, or that the problem simply defied categorisation.

As this example might suggest, it is usually not possible to consider the list of corrections we provide as being exhaustive, which raises questions for evaluation. We will discuss this point further in Section 4.

An additional twist here is that, for some of the edits in our data, a correction may 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), and so optional corrections are indicated as in the following example:

```
(9)  <edit type="RP">
        <original>; the</original>
        <corrections>
           <correction/>
           <correction>. The</correction>
        </corrections>
     </edit>
```

In this case, the first 'correction' suggests leaving the text as is, whereas the second suggests turning what was one sentence into two sentences. The first case is represented by an empty `<correction>` element.

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**.

3.2.5 Deletion and Insertion of Material

Two special circumstances we have to accommodate are as follows:

1. The original text may contain material that our annotators consider should be deleted, without any replacement text being provided.

2. The original text may be absent material that our annotators consider should be present, in which case there is no existing text that is being replaced.

These two cases are handled by edits like those in Examples (10) and (11) respectively:[9]

```
(10) <edit type="UD">
        <original>the␣</original>
        <corrections>
           <correction><empty/></correction>
        </corrections>
     </edit>
```

```
(11) <edit type="MD">
        <original><empty/></original>
        <corrections>
           <correction>the␣</correction>
        </corrections>
     </edit>
```

Each of these could be represented more simply, but we use the `<empty/>` tag in the first case for the sake of explicitness, and in the second case for the sake of consistency with a more complex situation, which we now describe.

It is possible that deletions and insertions may be optional, in which case we have the following representations corresponding to the two cases above:

```
(12) <edit type="UD">
        <original>the␣</original>
        <corrections>
```

[9]Note that in the case of deletion, the space following the word(s) to be removed must also be deleted; and in the case of insertion, the inserted material also needs to include a space. These spaces are indicated here by the '␣' character.

```
         <correction/>
         <correction><empty/></correction>
      </corrections>
   </edit>
```

(13)
```
<edit type="MD">
   <original><empty/></original>
   <corrections>
      <correction/>
      <correction>the␣</correction>
   </corrections>
</edit>
```

In each of these examples, the first correction is a null correction, meaning 'leave as is'; the second correction either deletes the identified text, or inserts text at the identified location.

3.2.6 Consistency Sets

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 csets is that of grammatical number agreement. In such a case, there are two possible corrections, but the items affected may be separated in the text, requiring two separate edits to be made. Suppose, for example, we have the following text:

(14) The item that we stored in the hash table are now made available.

This can be corrected by either making the noun *item* plural, or the verb *are* singular. We indicate these two possibilities by connecting the edits for each word via the `cset` attribute:[10]

(15)
```
<edit type="AGN" cset="1">
   <original>item</original>
   <corrections>
       <correction/>
       <correction>items</correction>
   </corrections>
</edit>

<edit type="AGV" cset="1">
   <original>are</original>
   <corrections>
      <correction>is</correction>
      <correction/>
   </corrections>
</edit>
```

In such cases, the convention is that the list of corrections should be related pairwise across consistency set elements: i.e., the first correction in the first edit corresponds to the first correction in the second edit, and so on.[11]

There are other more subtle cases where 'agreement' between corrections is required. A good example of this is the inverse of Example (9) given above. In that example, a semicolon was replaced by a full stop,

[10] This example demonstrates why it would be more sensible to have edit types associated with corrections rather than edits as a whole: the `AGN` and `AGV` types really correspond to the specific corrections made.

[11] Providing corrections with indices would make this more explicit, and also allow more flexibility in defining correspondences.

with a consequent change to the casing of the word following the punctuation mark, so that one sentence was broken into two. This was achieved via a single edit structure. But suppose instead that an annotator decides that an existing sentence boundary should be removed so that two sentences become one: this needs to be indicated via two edits that are connected via a `cset`, since an individual edit is not permitted to cross a sentence boundary. The following example demonstrates:

(16)
```
<edit type="RP" cset="1">
   <original>.</original>
   <corrections>
       <correction>;</correction>
   </corrections>
</edit>
<edit type="RP" cset="1">
   <original>The</original>
   <corrections>
      <correction>the</correction>
   </corrections>
</edit>
```

Note that this example retains whatever spacing was in the original text. In particular, if the original text followed the 'two spaces at the end of sentence' rule, and we prefer only one space after a semi-colon, then one or other of these edits would have to explicitly change the spacing.

3.2.7 Unspecified Corrections

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 is not clear what the missing text should be.
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, as in the following examples:

(17)
```
<edit type="MN">
   <original><empty/></original>
</edit>
```

(18)
```
<edit type="CE">
   <original>colourless green ideas sleep furiously</original>
</edit>
```

This approach supports the evaluation of system edits where participants still manage to offer corrections in such cases.

4 Evaluation

4.1 Why Evaluation is Hard

We need some form of evaluation to determine when the performance of a given system has improved, and to compare competing approaches in the Shared Task. However, 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.

Because of the above, any numbers that are computed as a means of characterising a system's performance need to be interpreted cautiously.

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. A little less convincingly, corrections might be thought as being analogous to the underlying semantics or identifiers associated with named-entity mentions: just as a named-entity mention may be ambiguous, and so have multiple possible referents, a markable may have multiple possible corrections. Of course, both named entities and markables as used here have associated types; however, in the first HOO round, we will not be evaluating the assignment of types to corrections.

Section 4.2 outlines the form of outputs that participating systems must produce for use as inputs to the evaluation process; Section 4.3 explains the details of how we score system outputs; and Section 4.4 provides a number of examples that demonstrate how scoring works, including some cases where it is clear that our current approach to evaluation does not do what we would prefer.

4.2 The Inputs to Evaluation

Each team will be assigned a two-character identification code, which should be incorporated into system output filenames to allow easy identification and tracking. Each team is also allowed to submit up to 10 distinct 'runs', so that they can provide alternative outputs. A single-digit number ranging from 0 to 9 will be used to identify a run; this will also be incorporated into output filenames. The format of a filename for a given fragment thus has the form shown in Example (19a), with an example shown in Example (19b):

(19) a. ⟨*FragmentID*⟩⟨*TeamID*⟩⟨*Run*⟩.⟨*FileType*⟩

b. **0067MQ5.txt**

The two permissible filetypes correspond to the two formats in which a team may provide data:

- For each input fragment, a correspondingly-named corrected text file may be provided; so, for team **MQ**, the first-run corrected output text for fragment 0001 would be named **0001MQ0.txt**.

- Alternatively, a team may provide for each input fragment a file containing XML edit structures in the same format as the gold-standard edits, with the restriction that only one correction per edit may be provided in each file. The corresponding output file would then be named **0001MQ0.xml**.

Teams thus have the choice of either creating a corrected version of the input texts, or of directly constructing a set of edit structures that correspond to their corrections.

- For the former case, we provide a tool that takes an original file and a corrected version of that file, and identifies the differences between these to produce a set of edit structures for use in evaluation.

- If a team decides to construct its own edit annotations, it is essential that these annotations be in exactly the same form as would be produced by our edit extraction tool.

As noted earlier, directly producing edit structures provides higher fidelity, which may in some situations provide better scores; instances of where this can happen are discussed below in Section 4.4. However, the need to track character offsets accurately makes the construction of edit structures more onerous than it might otherwise be. In the following, we will assume that the system output is in the form of a corrected text.

Suppose the original text is as shown in Example (20), and the corresponding gold-standard edit is as shown in Example (21):

(20)
```
The cat sit on the mat.
```

(21)
```
<edit type="TV" start="8" end="11">
   <original>sit</original>
   <corrections>
      <correction>sat</correction>
   </corrections>
</edit>
```

In order to match the gold standard, a participating system's output text would need to look as is shown in Example (22):

(22)
```
The cat sat on the mat.
```

Note that the system's output does not contain any annotations.[12] This has the consequence that the corrected texts cannot contain any indications of the hypothesized types of errors, and cannot indicate multiple alternative corrections. Although both of these are things we would eventually like to incorporate in our evaluations, for the pilot run our current intention is to keep things simple.

Given an original text file (containing text like that in Example (20)) and a corrected version of that file as provided by a participating system (containing text like that in Example (22)), our edit extraction utility will output an XML file that contains, in stand-off form, a collection of edit structures that capture the relevant information for each change found in the text. To enable evaluation, these stand-off annotations will include character offset information to enable identification of the specific text spans in the original file that they refer to. For the example above, the corresponding extracted edit will look like the following:[13]

(23)
```
<edit index="0067BT1-0001" start="8" end="11">
    <original>sit</original>
    <corrections>
       <correction>sat</correction>
    </corrections>
</edit>
```

Note that while the structure produced can accommodate multiple corrections, our desire to be able to accept system output consisting of plain unannotated text means that we impose a limit of one system-provided correction per edit. This restriction may be removed in subsequent rounds.

This example serves to demonstrate the format of edit structures that participating systems should adhere to if they construct these directly:

1. Every edit structure should have an index formed as shown in the example above.
2. Every edit structure should have start and end attributes that indicate the location of the markable in the original text.
3. The error type attribute does not need to be provided, since these are not evaluated in this round.
4. There should be at most one correction.
5. Care should be taken to ensure that spacing is properly incorporated when deleting or inserting material; see Section 3.2.5.

Evaluation then proceeds by comparing the set of gold-standard edit structures for a fragment with the set of edit structures corresponding to the participating team's output for that fragment.

[12] The text may contain sentence and paragraph tags as in the sentence-segmented source texts, but these will be ignored by the evaluation tools.

[13] The index value here is constructed by the extractor.

4.3 Scoring

There are a number of aspects of system performance for which we can derive scores:

- Detection: does the system determine that an edit is required at some point in the text?
- Recognition: does the system correctly determine the extent of the source text that requires editing?
- Correction: does the system offer a correction that is amongst the corrections provided in the gold standard?

For each of these, we can score a participating system on a fragment-by-fragment basis, or on a data set as a whole. We will also shortly release a version of the evaluation tools that assesses performance on individual error types across the data set as a whole.

In what follows, we will make use of the following terminology and notation:

- As noted above, each edit structure, whether in the gold standard or derived from a system output, indicates an extent by means of start and end offsets into the original text file. The edits for a given file are collected together into what we will call an **edit set**. We will indicate the set of gold-standard edits for a given input file as $G = \{g_1, \ldots, g_n\}$, and the set of system-produced edits for that file as $S = \{s_1, \ldots, s_m\}$. Note that the two edit sets to be compared may have different cardinality, since a system may not find all the spans to be corrected, or may identify spurious corrections.[14]
- Each edit structure has values for start and end offsets in the original text file. For some edit g_i, these are notated g_i*.start* and g_i*.end* respectively. For implementational efficiency, we will also assume that the elements of the sets are ordered in terms of increasing *start* values; this is a restriction that we may remove in later HOO rounds to enable more flexibility in evaluation.[15]

Given the above, we can define the notions of **strict alignment** and **lenient alignment**. Two edits g_i and s_j are said to be **strictly-aligned** if their start and end offsets are the same:

(24) $$(s_i\textit{.start} = g_j\textit{.start} \wedge s_i\textit{.end} = g_j\textit{.end})$$

The symbol '$\approx$' will be used to indicate strict alignment: for example, $(s_i \approx g_j)$.

Two edits g_i and s_j are said to be **leniently-aligned** if there is at least one character overlap:

(25) $$(s_i\textit{.start} \leq g_j\textit{.start} \wedge s_i\textit{.end} > g_j\textit{.start}) \vee (s_i\textit{.start} < g_j\textit{.end} \wedge s_i\textit{.end} \geq g_j\textit{.end})$$

The symbol '$\sim$' will be used to indicate lenient alignment: for example, $(s_i \sim g_j)$. In an ideal case, of course, edits in the two edit sets will correspond one-to-one, but this may not always be the case.

For any given pair of edit sets $\langle G, S \rangle$, an **alignment set** is a structure that indicates the correspondences between the edits in the two edit sets. We have two alignment sets for any pair of edit sets: a **strict alignment set**, containing just the strict alignments, and a **lenient alignment set**, containing the lenient alignments. Every strict alignment will also be represented in the lenient alignment set, but not vice versa.

We also have what we call **unaligned edits**. These are edits which do not appear in the lenient alignment set. An unaligned system edit corresponds to a **spurious** edit; an unaligned gold-standard edit corresponds to a **missing** edit. It is important to note that missing edits are of two types, depending on whether the gold-standard edit corresponds to an optional edit or a mandatory edit (see Section 3.2.4). 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 need to keep track of the number of **missing optional edits**.

In what follows, we will use the situation shown schematically in Figure 6 to demonstrate these concepts. The various sets of edits shown here are as follows (assuming for present purposes that all of the gold-standard edits shown are mandatory):

- The gold-standard edit set $G = \{g_1, g_2, g_3, g_4, g_5\}$

[14] At least, spurious from the point of view of the gold standard. Of course, there may be quite appropriate edits that are missing from the gold standard.

[15] Note that in the current setup, no two edits in an edit set can have the same *start* value.

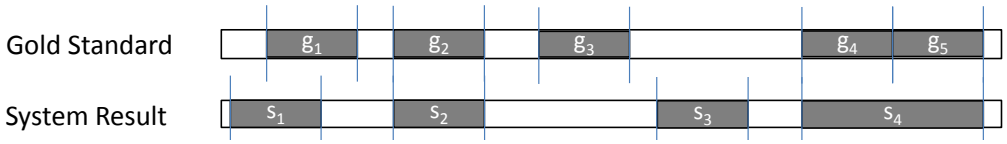

Figure 6: Example Alignments

- The system-produced edit set $S = \{s_1, s_2, s_3, s_4\}$
- The strict alignment set = $\{(s_2 \approx g_2)\}$
- The lenient alignment set = $\{(s_1 \sim g_1), (s_2 \sim g_2), (s_4 \sim g_4), (s_4 \sim g_5)\}$
- Missing mandatory edits = $\{g_3\}$
- Missing optional edits = $\{\}$
- Spurious edits = $\{s_3\}$

We will use this example in what follows to show how scoring works.

4.3.1 Detection

For a given $\langle G, S \rangle$ pair, a gold standard edit g_i is considered **detected** if there is at least one alignment in the lenient alignment set that contains g_i. We then calculate Precision and Recall as follows:[16]

$$Precision = \frac{\#\ of\ detected\ edits}{\#\ of\ spurious\ edits + \#\ of\ detected\ edits} \tag{26}$$

$$Recall = \frac{\#\ of\ detected\ edits}{\#\ of\ gold\text{-}standard\ edits - \#\ of\ missing\ optional\ edits} \tag{27}$$

Note that the number of detected edits is not necessarily the size of the lenient alignment set. If there are multiple lenient alignments that contain g_i, then these count fractionally towards the score: so, if there are n lenient alignments containing g_i then each counts as $\frac{1}{n}$ detections; alternatively, only the first lenient alignment containing g_i needs to be counted.

The detection score for a given $\langle G, S \rangle$ pair is then the harmonic mean (F-score):

$$DetectionScore = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{28}$$

This score is calculated for each document; for a system's performance across all documents in a data set, we compute the average of the scores across all the documents. We thus have **per-document detection scores** and **dataset detection scores**.

In the example in Figure 6, we have four detected edits (g_1, g_2, g_3, and g_4) and one spurious edit (s_3), so the Precision is 0.8 and the Recall is also 0.8, for a *DetectionScore* of 0.8. If g_3 was an optional edit, then the Recall would instead be 1, for a *DetectionScore* of $0.8\dot{8}$.

A trickier case, which we refer to as **staggered alignment**, is shown schematically in Figure 7. Here, the following alignments hold:

$$\{(s_1 \sim g_1), (s_2 \sim g_1), (s_2 \sim g_2)\} \tag{29}$$

In this case, we have two detected edits (g_1 and g_2); although there are three alignments in the lenient alignment set, two of these correspond to the same g_i and thus only count for a half each. There are no spurious edits, so the Precision is 1, and the Recall is also 1, for a *DetectionScore* of 1.

[16]Note that in all computations of Precision and Recall 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.

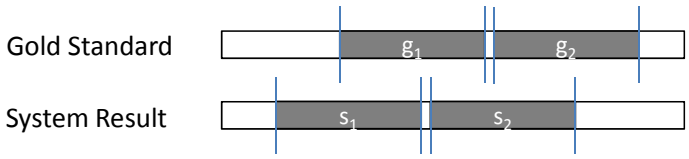

Figure 7: Staggered Alignment

4.3.2 Recognition

The detection score described above can be considered a form of 'lenient' recognition. We also want to measure 'strict' recognition, i.e. the degree to which a participating system is able to determine the correct start and end locations of text to be corrected. We consider a gold-standard edit g_j 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 have:

$$Precision = \frac{\#\ of\ recognized\ edits}{\#\ of\ system\ edits} \tag{30}$$

$$Recall = \frac{\#\ of\ recognized\ edits}{\#\ of\ gold\text{-}standard\ edits - \#\ of\ missing\ optional\ edits} \tag{31}$$

The recognition score for a given $\langle G, S \rangle$ pair is then the harmonic mean (F-score):

$$RecognitionScore = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{32}$$

Again, the recognition score can be calculated for each document, and for the set of documents in the data set.

In the example in Figure 6, we have one recognized edit (g_2), four system edits, and five gold-standard edits, so the Precision is 0.25 and the Recall is 0.2, for a *RecognitionScore* of $0.2\dot{2}$. If g_3 was an optional edit, then the Recall would instead be 0.25, for a *RecognitionScore* of 0.25.

This case demonstrates a deficiency in the scoring scheme: note that one system edit (s_4) corresponds to the same extent as two gold-standard edits (g_4 and g_5), so it is quite possible that the system has decomposed what the gold-standard sees as a single edit into two constituent edits. Both analyses may be plausible; however, the scoring scheme gives no recognition credit in this case.

In the staggered alignment example of Figure 7, we have no recognized edits, so the *RecognitionScore* is 0.

4.3.3 Correction

Recall that for any given gold-standard edit g_j, there may be multiple possible corrections. A system edit s_i 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 have:

$$Precision = \frac{\#\ of\ valid\ corrections}{\#\ of\ system\ edits} \tag{33}$$

$$Recall = \frac{\#\ of\ valid\ corrections}{\#\ of\ gold\text{-}standard\ edits - \#\ of\ missing\ optional\ edits} \tag{34}$$

The correction score for a given $\langle G, S \rangle$ pair is then the harmonic mean (F-score):

$$\textit{CorrectionScore} = 2 \times \frac{\textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}} \tag{35}$$

Again, the correction score can be calculated for each document, and for the set of documents in the data set.

In Figure 6, if s_2 is a valid correction for g_2 (the only candidate since this is the only strict alignment), then the *CorrectionScore* is $0.2\dot{2}$. If every recognition corresponds to a valid correction, then the *CorrectionScore* will be the same as the *RecognitionScore*. If only a proportion of the recognitions correspond to valid corrections, then the *CorrectionScore* will be proportionately reduced. The *CorrectionScore* can never be greater than the *RecognitionScore*.

4.4 Examples and Test Cases

In this section we iterate through a number of examples to how scoring works in each case. To simplify the presentation, we introduce a shorthand notation for showing the content of the gold-standard edits inline: the markable and its corrections are bounded by braces, with the markable and the set of corrections separated by a '$\rightarrow$' character, and each pair of possible corrections separated by a '$\wedge$' character. A leading '$\wedge$' in the set of corrections indicates that the correction is optional (i.e., the first alternative correction is null), so that it is valid to leave the text as it is. Changes present in the system output will be underlined.

In most cases we will assume that the gold standard only contains one possible correction for each markable.

4.4.1 Perfect Match

Here there is a strong alignment that provides a valid correction.

(36) Original Text: The cat sit on the mat.
Gold Standard: The cat {sit→sat} on the mat.
System Output: The cat <u>sat</u> on the mat.

If the text shown was the complete fragment, then:

- *DetectionScore* = 1
- *RecognitionScore* = 1
- *CorrectionScore* = 1

4.4.2 Failure to Detect

In this case, we have a missing mandatory edit.

(37) Original Text: The cat sit on the mat.
Gold Standard: The cat {sit→sat} on the mat.
System Output: The cat sit on the mat.

If the text shown was the complete fragment, then:

- *DetectionScore* = 0
- *RecognitionScore* = 0
- *CorrectionScore* = 0

4.4.3 Spurious Correction

In this case, the system proposes an edit (here, *on* → *at*) which does not have a corresponding edit in the gold standard.

(38) Original Text: The cat sit on the mat.
Gold Standard: The cat {sit→sat} on the mat.
System Output: The cat sit at the mat.

Once again, if the text shown was the complete fragment, then:

- *DetectionScore* = 0
- *RecognitionScore* = 0
- *CorrectionScore* = 0

This means there is no penalty assigned for spurious corrections, which might be considered a weakness in the current scoring scheme. We could address this simply by keeping count of the number of spurious corrections a system makes.

4.4.4 Invalid Correction

Here the system proposes a correction which is not amongst those offered in the gold standard, although the need for a correction is correctly recognised.

(39) Original Text: The cat sit on the mat.
Gold Standard: The cat {sit→sat} on the mat.
System Output: The cat sits on the mat.

If the text shown was the complete fragment, then:

- *DetectionScore* = 1
- *RecognitionScore* = 1
- *CorrectionScore* = 0

As already noted, it's possible that a system might propose a correction that is quite acceptable, but was simply not foreseen by the annotators, so the system will be unfairly penalised.

4.4.5 Multiple Possible Corrections

In this example, the gold standard proposes multiple possible corrections. Provided the system offers one of the proposed corrections, this is no different from the Perfect Match case described above.

(40) Original Text: The cat sit on the mat.
Gold Standard: The cat {sit→sat∧sits} on the mat.
System Output: The cat sits on the mat.

If the text shown was the complete fragment, then:

- *DetectionScore* = 1
- *RecognitionScore* = 1
- *CorrectionScore* = 1

4.4.6 Optional Correction

In this situation, the gold standard offers a correction, but also allows the possibility that the text may be left as it is in the original. First suppose the system does not propose a correction, so no edits are detected.

(41) Original Text: The cat sat on the mat.
Gold Standard: The cat sat {on→ ∧at} the mat.
System Output: The cat sat on the mat.

If the text shown was the complete fragment, then:

- *DetectionScore* = 1
- *RecognitionScore* = 1
- *CorrectionScore* = 1

On the other hand, suppose the system does offer a correction in such a case, but it is not one of the corrections offered by the system:

(42) Original Text: The cat sat at the mat.
Gold Standard: The cat sat {on→ ∧at} the mat.
System Output: The cat sat in the mat.

Then we have:

- *DetectionScore* = 1
- *RecognitionScore* = 1
- *CorrectionScore* = 0

Again, it is entirely possible that the system correction is in fact a plausible correction that was not captured in the gold standard.

4.4.7 Extent Generalisation

We indicated in Section 3.2.2 that the extent of a markable in the original text is determined by the longest correction provided in the gold standard. This has an impact on scoring. To repeat the example from earlier, suppose we have the following possible corrections for the original text shown:

(43) Original Text: ...the ambiguity of the person name ...
Correction #1: ...the ambiguity of the personal name ...
Correction #2: ...the ambiguity of the person's name ...
Correction #3: ...the ambiguity of personal names ...

The gold standard edit representing these corrections looks like the following:

(44)
```
<edit type="R" start="17" end="34">
   <original>the person name</original>
   <corrections>
      <correction>the personal name</correction>
      <correction>the person's name</correction>
      <correction>personal names</correction>
   </corrections>
</edit>
```

Suppose a participating system proposes the following edit, which is equivalent to the first alternative in the gold standard in terms of its actual effect on the text:

(45)
```
<edit type="R" start="21" end="29">
   <original>person</original>
   <corrections>
      <correction>personal</correction>
   </corrections>
</edit>
```

Then the scores would be as follows

- *DetectionScore* = 1
- *RecognitionScore* = 0
- *CorrectionScore* = 0

Thus, the system has produced a correction whose net effect is the same as one of the alternatives in the gold standard, but it receives no recognition or correction credit for this because the extent does not match that in the gold standard.

4.4.8 Edit Aggregation

Another problematic situation arises when a participating system identifies one markable that is co-extensive with a series of markables in the gold standard. Consider the following fragment and its possible correction:

(46) Original Text: The cat sit at the mat.
Corrected Text: The cat sat on the mat.

The problem here is that the correction can be seen as consisting of either one edit or two edits. Suppose the gold standard annotates this as two distinct edits:

(47)
```
<edit type="FV" start="8" end="11">
   <original>sit</original>
   <corrections>
      <correction>sat</correction>
   </corrections>
</edit>

<edit type="RT" start="12" end="14">
   <original>at</original>
   <corrections>
      <correction>on</correction>
   </corrections>
</edit>
```

Now suppose a participating system combines these into a single edit:

(48)
```
<edits>
  <edit start="8" end="14">
    <original>sit at</original>
    <correction>sat on</correction>
  </edit>
</edits>
```

Consequently, although the system has properly corrected the text, it will only receive credit via lenient extent recognition, and will receive no credit for recognition or correction:

- *DetectionScore* = 1
- *RecognitionScore* = 0

- *CorrectionScore* = 0

Note that this problem can arise artificially as a result of using our edit extraction tool even if the participating system had actually detected the two edits separately: if the edits are adjacent in the text, then they will be extracted as a single edit. This is one reason for constructing the edit set directly, rather than relying on our tool.

4.4.9 Edit Decomposition

Another problematic case is the opposite of that just described. Suppose, for the same text in Example 46, we now have one edit in the gold standard, but the participating system proposes two edits that are together co-extensive with the gold standard edit. Then:

- *DetectionScore* = 1
- *RecognitionScore* = 0
- *CorrectionScore* = 0

Again, the system produces the right result, but is not given credit for either recognition or the content of the correction.

4.5 Summary

In this section, we've described the approach we're taking to evaluation in HOO. Although, as we have noted, there are some the potential weaknesses in the current scoring regime, we believe it provides a good starting point which can be refined on the basis of experience. The evaluation scripts we provide are described in Appendix D.

5 Conclusions and Outstanding Issues

This report has described the nature of the annotations used in the initial HOO data set, and explained the reasoning behind a number of design decisions that have been taken. We've also explained the approach we will be taking to evaluation.

Appendix F provides a list of known issues and possible improvements to the framework described here. However, as noted at the start of this document, we see the pilot round of HOO as one where we learn from mistakes, and determine better answers to some of the tricky questions. Accordingly, we welcome community input on any aspect of the exercise, and in particular on the three following topics:

The Annotation Scheme: We believe our existing approach to annotation covers all the necessary cases, but we welcome feedback on any aspect of the design of the annotation; we expect that a number of revisions will be made for subsequent HOO rounds, some of which are foreshadowed in comments throughout this document.

The Annotated Data: The data set for the pilot HOO round has now been finalised, with a number of minor revisions and corrections having been integrated based on feedback from the community in response to the preliminary release. We do not expect to make any more changes to the data for this round, but we would still like to be informed of any errors that are found, so that these can be corrected for subsequent use.

Evaluation Methodology: We would appreciate any input on the approach we are taking to evaluation, and any views on the unresolved issues that remain.

We invite feedback via the HOO Google Group at `http://groups.google.com/group/hoo-nlp`. Up-to-date information about the Shared Task can always be found at `www.clt.mq.edu.au/research/projects/hoo`.

6 Acknowledgements

Diane Nicholls and Kate Wild diligently annotated the data for the pilot phase. Guy Lapalme provided useful input on XML, XSLT and CSS; Daniel Dahlmeier identified some tricky cases for evaluation; and Joel Tetrault provided useful feedback on a draft of the evaluation plan.

References

Dale, R. 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.

Nicholls, D. 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.

A The Cambridge Learner Corpus Error Tags

The table below shows the error codes used in the manual annotation. This error coding system has been developed by Cambridge University Press, with the exception of one code, CC (for Compound Change; see Example (8) in Section 3.2.4), which has been added for the present project. Some of the tags listed below are *not* currently used in HOO; in particular, instances of US spelling (type SA) are not considered errors. Note that the error tags used have a different syntax when used by Cambridge University Press to that used here.

The Cambridge University Press Error Coding System is copyright to Cambridge University Press and may only be used with their written permission. The coding is used to annotate the Cambridge Learner Corpus, which informs English Language Teaching materials published by Cambridge University Press. The scheme is discussed in detail in (Nicholls, 2003).

Tag	Meaning	Tag	Meaning
AG	agreement error	MA	missing anaphor
AGA	anaphor agreement error	MC	missing link word
AGD	determiner agreement error	MD	missing determiner
AGN	noun agreement error	MJ	missing adjective
AGV	verb agreement error	MN	missing noun
AS	argument structure error	MP	missing punctuation
C	countability error	MQ	missing quantifier
CD	wrong determiner because of noun countability	MT	missing preposition
CE	complex error	MV	missing verb
CL	collocation or tautology error	MY	missing adverb
CN	countability of noun error	NE	no error
CQ	wrong quantifier because of noun countability	R	replace error
DA	derivation of anaphor error	RA	replace anaphor
DC	derivation of link word error	RC	replace link word
DD	derivation of determiner error	RD	replace determiner
DI	incorrect determiner inflection	RJ	replace adjective
DJ	derivation of adjective error	RN	replace noun
DN	derivation of noun error	RP	replace punctuation
DQ	derivation of quantifier error	RQ	replace quantifier
DT	derivation of preposition error	RT	replace preposition
DV	derivation of verb error	RV	replace verb
DY	derivation of adverb error	RY	replace adverb
FA	wrong anaphor form	S	spelling error
FC	wrong link word form	SA	spelling American
FD	incorrect determiner form	SX	spelling confusion
FJ	wrong adjective form	TV	incorrect tense of verb
FN	wrong noun form	U	unnecessary error
FQ	wrong quantifier form	UA	unnecessary anaphor
FT	wrong preposition form	UC	unnecessary link word
FV	wrong verb form	UD	unnecessary determiner
FY	wrong adverb form	UJ	unnecessary adjective
IA	incorrect anaphor inflection	UN	unnecessary noun
ID	idiom wrong	UP	unnecessary punctuation
IJ	incorrect adjective inflection	UQ	unnecessary quantifier
IN	incorrect noun inflection	UT	unnecessary preposition
IV	incorrect verb inflection	UV	unnecessary verb
IY	incorrect adverb inflection	UY	unnecessary adverb
L	inappropriate register	W	word order error
M	missing error	X	incorrect negative formation

B Error Codes by Group with Examples

The examples shown here have been invented solely to illustrate clearly the error codes, and are reproduced with the kind permission of Cambridge University Press. They are not taken from actual learner writing.

Verb errors

Error Code	Description	Example
RV	Replace verb	I existed last weekend in London
TV	Wrong tense of verb	I spend last weekend in London
FV	Wrong verb form	I to spend last weekend in London
MV	Missing verb	I last weekend in London
UV	Unnecessary verb	I spent to be last weekend in London
IV	Incorrect inflection of verb	I spended last weekend in London
DV	Derivation of verb error	I spendified last weekend in London
AGV	Verb agreement error	The three birds is singing

Noun errors

Error Code	Description	Example
RN	Replace noun	Have a good travel!
FN	Wrong noun form	I met the ladies
IN	Wrong noun inflection	I met two ladys
MN	Missing noun	It was an interesting
UN	Unnecessary noun	He has a huge office room
CN	Countability error	I dont have any monies
DN	Derivation of noun error	There are changements to the schedule
AGN	Noun agreement error	One of my friend

Adjective errors

Error Code	Description	Example
RJ	Replace adjective	The afternoon was very bored
FJ	Form of adjective	The situation got worst
MJ	Missing adjective	Kindly reply as soon as
IJ	Wrong adjective inflection	My news shoes
DJ	Derivation of adjective error	An interessant story
UJ	Unnecessary adjective	It was 3 o'clock in the early morning

Adverb errors

Error Code	Description	Example
RY	Replace adverb	He stared at her intensively
FY	Form of adverb	As I said early, . . .
MY	Missing adverb	He pointed my mistake
UY	Unnecessary adverb	I went together with him
DY	Derivation of adverb error	It happened fastly
IY	Inflection of adverb error	Please drive slowlier

Preposition errors

Error Code	Description	Example
RT	Replace preposition	When I arrived at London
MT	Missing preposition	I gave it John
UT	Unnecessary preposition	I told to John that . . .
DT	Derivation of preposition	He looks alike his father

Conjunction errors

Error Code	Description	Example
RC	Replace Conjunction	I wonder and he will come
UC	Unnecessary conjunction	But although I dont know.
MC	Missing conjunction	The stripes were red green

Quantifier errors

Error Code	Description	Example
FQ	Form of quantifier	It is one of the fewer things I like
RQ	Replace quantifier	There were people of any age there
MQ	Missing quantifier	Ill call in the next days
UQ	Unnecessary quantifier	A little bit quite common
CQ	Quantifier countability error	It cost him many money
AGQ	Quantifier agreement error	In another circumstances
DQ	Derivation of quantifier	I earn lesser money in this job
IQ	Inflection or quantifier error	I enjoy chess and others games

Anaphor/pronoun errors

Error Code	Description	Example
FA	Form of anaphor	To who it may concern
RA	Replace anaphor	I have a car. She is blue
MA	Missing anaphor	I have a car. Is blue
UA	Unnecessary anaphor	My car it is blue
DA	Derivation of anaphor	Yous sincerely
AGA	Anaphor agreement error	It were difficult questions
IA	Inflection of anaphor	Thank you for everythings

Determiner errors

Error Code	Description	Example
FD	Form or determiner	I have an car
RD	Replace determiner	Have the nice day
MD	Missing determiner	I have car
UD	Unnecessary determiner	There was a lot of the traffic
DD	Derivation of determiner	Shes name was Anna
AGD	Determiner agreement error	I enjoy these job
CD	Countability of determiner	I hope these news reaches you
DI	Inflection of determiner	Thank you for yours letter

Punctuation errors

Error Code	Description	Example
RP	Replace punctuation	The womans handbag
MP	Missing punctuation	The womans handbag
UP	Unnecessary punctuation	The womans hand-bag

Spelling errors

Error Code	Description	Example
S	Wrong spelling	accomodation
SX	Spelling confusion	I'll see you their
SA	US spelling	color

Collocation errors

Error Code	Description	Example
CL	Collocation error	She beat me blue and black
CL	Tautology error	Im pleased and happy to be here

Register errors

Error Code	Description	Example
L	Register error (Label)	A chum of mine informed me

Negative formation errors

Error Code	Description	Example
X	Negative formation error	Love dont live here no more

Complex errors

Error Code	Description	Example
CE	Complex error	He didnt never should be having

Idiom errors

Error Code	Description	Example
ID	Idiom error	In one hand . . . In another hand

Argument structure errors

Error Code	Description	Example
AS	Argument structure error	It gives great pleasure to me

Word order errors

Error Code	Description	Example
W	Word order	I have also two cats
W	Word order (possessive)	The hat of the man

C Statistics

The initial data set consists of 19 fragments selected from papers previously published in ACL conferences and workshops, and used with the kind permission of the original authors. Each fragment is approximately 1000 words long; the precise word counts for each fragment are as shown in Table 1.

Table 2 shows the number of errors of each type contained in the initial data set as a whole. Within the 19 files, 888 edits offer one possible correction; 320 offer two corrections; 27 offer three corrections; and two offer four corrections. In 76 cases, correction is optional.

File	Word Count	File	Word Count
0001.txt	1005	0044.txt	1141
0004.txt	1004	0046.txt	1009
0005.txt	1013	0048.txt	1026
0015.txt	1022	0049.txt	1033
0019.txt	1061	0054.txt	1033
0020.txt	1048	0055.txt	1013
0021.txt	1100	0061.txt	1009
0031.txt	996	0062.txt	1064
0038.txt	1063	0066.txt	953
0041.txt	1008		

Table 1: Word Counts in the Initial Data Set

Type	Count	Type	Count	Type	Count
AGD	4	AGN	17	AGV	14
AS	2	CC	93	CE	2
CN	1	CQ	2	DJ	5
DN	9	DQ	1	DY	6
FD	3	FJ	2	FN	46
FV	4	ID	1	IJ	2
IN	2	L	5	M	14
MA	2	MC	5	MD	161
MN	9	MP	210	MT	5
MV	11	MY	4	R	84
RA	3	RC	4	RD	22
RJ	36	RN	67	RP	67
RQ	3	RT	110	RV	64
RY	19	S	11	TV	22
U	3	UA	1	UC	3
UD	33	UJ	1	UN	2
UP	17	UT	15	UV	3
UY	4	W	29		

Table 2: Errors by Type

D Using the Evaluation Tools

D.1 Scores To Be Reported

Given a collection of test data set consisting of fragments $T_1 \ldots T_n$, we compute the following:

1. Per-fragment scores:
   (a) *DetectionScore* for each T_i
   (b) *RecognitionScore* for each T_i
   (c) *CorrectionScore* for each T_i
2. Average scores across the data set as a whole:
   (a) *DetectionScore* averaged across all T_i
   (b) *RecognitionScore* averaged across all T_i
   (c) *CorrectionScore* averaged across all T_i

D.2 Tools Provided

To support evaluation, we provide two Python scripts:

batch_evaluate.py: Computes *DetectionScore*, *RecognitionScore* and *CorrectionScore* for all fragments in a directory, and reports result for each fragment as well as the averages over all fragments; the output is in .csv format.

evaluate.py: Computes *DetectionScore*, *RecognitionScore* and *CorrectionScore* for a single fragment.

These operate on files that contain edit structures, and can be used directly by teams who create their own edit structures as output.

For those who create corrected text files as output, we also provide software for extracting edit structures from these corrected text files:

batch_extract.py: given a directory containing a set of original text files and a second directory containing a set of corrected files, produces in a third specified directory a set of files containing the edit structures that correspond to the changes in the set of corrected files.

batch_extract.py uses **diffextract.py**, which is included in the package, and Gnu **wdiff**, which you will need to install separately.[17]

D.3 Running the Tools

Evaluation is carried out by comparing a set of system edits against a set of gold standard edits. If your system outputs corrected texts, start at Section D.3.1; if you directly construct edit structures, jump to Section D.3.2.

D.3.1 Extracting Edits from Corrected Text

If your system outputs corrected text files, you need to generate a set of system edits from the corrected text files. This is done using the **batch_extract.py** script. Here we assume three directories:

Orig: A directory containing a set of original text fragments.

System: A directory containing a set of system outputs, these being corrected versions of the original text fragments.

[17] This can be found at `http://www.gnu.org/software/wdiff`. If you are a Windows user, **wdiff** is also available as a package for cygwin: see `http://cygwin.com`. Note that wdiff is not part of the base cygwin installation, and therefore has to be installed separately.

File	detectionprecision	detectionrecall	detectionscore	recognitionprecision	recognitionrecall	recognitionscore	correctionprecision	correctionrecall	correctionscore
0441MQ1	1	1	1	1	1	1	1	1	1
0442MQ1	0	0	0	0	0	0	0	0	0
0443MQ1	0	0	0	0	0	0	0	0	0
0444MQ1	1	1	1	1	1	1	0	0	0
0445MQ1	1	1	1	1	1	1	1	1	1
0446MQ1	1	1	1	1	1	1	0	0	0
0447MQ1	1	1	1	0	0	0	0	0	0
0448MQ1	1	1	1	0	0	0	0	0	0
Average	0.75	0.75	0.75	0.5	0.5	0.5	0.25	0.25	0.25

Figure 8: Evaluation results

SysXML: A directory that will be used to contain the output of edit structure extraction from the system output texts.

To extract the edits, proceed as follows:

```
> python batch_extract.py Orig System SysXML
Compiling files...
2 original files found in Orig
2 system files found in System

Matching system files to original files...
2 pairs found
Extract diffs...
Finished.
>
```

D.3.2 Evaluating Edit Sets

Here we assume two directories:

GoldXML: A directory containing the gold-standard edit structures corresponding to the corrected versions of the original text fragments.

SysXML: A directory containing the corresponding edit structures produced by the participating system.

Run **batch_evaluate.py** to produce the evaluation results:

```
> python batch_evaluate.py GoldXML SysXML Results.csv
Compiling files...
8 gold files found in GoldXML
8 system files found in SysXML

Matching system files to gold files...
8 pairs found

Evaluating system output...
Opening Results.csv for score output
Scores successfully written to Results.csv
>
```

The results of a sample evaluation run are shown in Figure 8.[18]

More detailed results can be obtained by running **evaluate.py** on a single pair of files:

[18]This uses a set of sample files provided with the code, which correspond to the test cases discussed in Section 4.4.

```
> python evaluate.py 0446G.xml 0446MQ1.xml
Parsing edits in 0446G.xml...
Found 1 edits
Parsing edits in 0446MQ1.xml...
Found 1 edits

Scoring detection...
  Detected edits: 1
  Missed optional edits: 0
  Spurious edits: 0
  Gold edits: 1
- Precision: 1 / (0 + 1) = 1.00
- Recall: 1 / (1 - 0) = 1.00
- Score: 1.00

Scoring recognition...
  Recognised edits: 1
  Missed optional edits: 0
  Gold edits: 1
  System edits: 1
- Precision: 1 / 1 = 1.00
- Recall: 1 / (1 - 0) = 1.00
- Score: 1.00

Scoring corrections...
  Aligned system edits: 1
  Valid corrections: 0
  Gold edits: 1
  System edit: 1
- Precision: 0 / 1 = 0.00
- Recall: 0 / 1 = 0.00
- Score: 0.00

detection
  recall: 1.00
  score: 1.00
  precision: 1.00

recognition
  recall: 1.00
  score: 1.00
  precision: 1.00

correction
  recall: 0.00
  score: 0.00
  precision: 0.00
End.
>
```

E Glossary

This appendix provides a list of the terms used in this document, along with their definitions.

consistency set: a set of edits which are interdependent.

correction: text that is offered as a replacement for a markable in a fragment.

data set: a collection of fragments corresponding to a training or testing set.

detection: determining that an edit is required at some point in a text.

edit: an indication of a change to a markable in a fragment, typically specified as one or more possible corrections for that markable.

edit, gold-standard: an edit whose contents define the 'correct answer' for a given markable.

edit, mandatory: a gold-standard edit which is required; i.e., one of the provided corrections should be applied.

edit, missing: an edit in the gold standard for which a participating system does not propose an aligned edit.

edit, optional: a gold-standard edit which is not a mandatory edit; i.e., the text can be left as is (one of the corrections is a null correction).

edit, spurious: an edit proposed by a participating system for which there is no aligned edit in the gold standard.

edit structure: an XML object that contains the information defining the possible corrections for a given markable.

edit set: a collection of edits corresponding to a fragment.

extent: a span of text defined by start and end character positions in a fragment.

fragment: an excerpt of a text, corresponding to one file in a **data set**; identified by a four-digit number, and typically containing around 1000 words of text.

lenient alignment: the situation that obtains when the extent of an edit proposed by a system overlaps by at least one character the extent of an edit contained in the gold standard.

markable: a span of text that is considered correctable.

recognition: determining the extent of an edit.

run: a set of output files generated by a participating team for a given data set under some software configuration.

source document: the original document from which a fragment has been drawn.

strict alignment: the situation that obtains when an edit proposed by a system refers to the same extent as an edit contained in the gold standard.

team ID: a two-character code used in filenames and edit indices to identify a participating team.

unaligned edit: when comparing two set of edits, any edit which is not aligned with an edit in the other set.

F Known Issues and Possible Improvements

Here's a list of things we know could do with fixing, or at least are worth thinking more about. If you want to report a problem or possible issue, please first check that it isn't already in this list.

F.1 Source Text Annotation

1. The sentence-segmented version of the data contains only `<p>` and `<s>` tags. A consequence of this is that some text categories which are not sentences, such as items in lists, are tagged as sentences. In future versions, it may be useful to use other tags to indicate headings, list items and other distinct elements of logical structure.
2. It may be useful to add index attributes to `<p>` and `<s>` elements so that they can be uniquely identified.
3. Our use of plain text as the standard format for data means that we are not able to correct typographic problems such as incorrect or dispreferred use of italics and boldface; also, we can't handle text that contains mathematical symbols.

F.2 Gold Standard Data Issues

1. Data may contain corrections to spellings where the form corrected could be considered valid. So, for example, file 0055G.xml identifies *modelling* and *modelled* as misspelled words, but they may be considered acceptable variants.

F.3 Edit Structure Format

1. Currently, error types are attributes of `edit` elements. However, error types should probably be associated with corrections rather than edits as a whole, since it is conceivable that different corrections may correspond to different perceptions of what the nature of the error is.
2. Each correction within an edit could usefully have an identifying index.

F.4 Supporting Tools and Documentation

1. Modify the XSLT and CSS support to enable browsing of inline annotated data in Internet Explorer.
2. At least one valid error tag, `AGQ` (quantifier agreement), is missing from the table in Appendix A.

F.5 Evaluation

1. Modify scoring regime to give partial marks depending on the degree of overlap, rather than the current binary correct vs incorrect.
2. Keep track of and report on the number of spurious edits that systems propose.
3. Other forms of evaluation: classify a sentence as containing an error or not; use BLEU or edit distance to compare original and corrected texts.