ErrorIST: towards the automatic evaluation of editors

Tiago Santos, Luísa Coheur
Spoken Language Systems Lab (L2F)
INESC-ID
Av. Prof. Doutor Cavaco Silva
2744-016 Porto Salvo
tiago.f.dos.santos@tecnico.ulisboa.pt
luisa.coheur@inesc-id.pt

João Graça
Unbabel
Rua Visconde de Santarém, 67B
1000-286 Lisboa, Portugal

Abstract

In this paper, we present a configurable tool, ERRORIST, capable of creating error prone text and verifying it for corrections. ErrorIST is an extensible tool, capable of inserting a variety of errors based on a translation error taxonomy. Artificial error insertion is configurable on error type variety. ERRORIST focuses on the insertion of errors in European Portuguese. We evaluate this tool by submitting its errors to editor correction and detecting their corrections automatically. Evaluating the corrections accurately is a challenge and while ERRORIST cannot completely replace manual evaluation, it creates the evaluation material reliably and fast whilst reducing the need for human verification in correction detection. ERRORIST can reduce the need for manual correction verification in 66.83%, even further (70.48%) if you consider non-translation editing.

1 Introduction

A tool capable of adding different types of errors to a sentence can be helpful in several scenarios. For instance, such tool can be used to generate parallel corpora (correct sentences vs. sentences with errors), from which a model capable of "translating" incorrect sentences into correct ones can be built, as done in (Brockett et al., 2006). Also, such tool can be used to evaluate someone's proficiency in a language, by providing texts with errors that need to be corrected. The way the person being tested is able to correct those errors can be an indicator of his/her skills in that language. In this paper, we focus in this latter scenario, where different types of errors, automatically generated by a framework called ERRORIST, are given to someone (from now on, the editor) that should correct them. ERRORIST will then evaluate his/her performance, considering the editions made.

Contrary to other systems that perform general error insertion, as, for instance, GenERRate (Foster and Andersen, 2009), ERRORIST follows a fine-grained taxonomy of errors, described in (Costa et al., 2015). As a consequence, ERRORIST allows the introduction of very specific error types (capitalization, addition of functional or content words, errors regarding contractions, just to name a few), and, thus, to perform a precise evaluation of the editor. Although we focus on the Portuguese language, most of ERRORIST errors rely on resources available for many other languages. For instance, ERRORIST takes advantage of Ispell affix files, available for 50 languages[1].

Unfortunately, to be able to insert errors into texts is not enough to automatically evaluate an editor: his/her editions need to be traced. This process presents a challenge that results from the fact that the editor can correct parts of the sentence in which no error was added. In this way, a correct sentence, different from the expected one, can be returned by the editor, and no conclusion can be taken regarding his/her proficiency. For instance, taking into account the sentence *"The kids are playing in the yard."*, and the same sentence with an error in the verb[2] *"The kids are **play** in the yard."*, the editor can correct it by returning *"The kids play in the yard."*, which is also correct and not the "expected" correction. In this paper, besides introducing and evaluating ERRORIST, we present a study that relates each error type with its traceability. This is due to the fact that some errors are almost 100% traceable and others are not.

[1] https://www.gnu.org/software/ispell/ispell.html

[2] For notational purposes, the generated errors will appear between **, and if an error results from a word that is missing, we will use [ ].

This paper is organized as follows: in Section 2 we present ERRORIST general architecture, and, in Section 3, we describe, in detail, the errors implemented in ERRORIST. In Section 4, we evaluate ERRORIST and discuss how traceable each error type is. In Section 5 we present some related work. Finally, in Section 6 we present the main conclusions and point to future work.

2 ERRORIST's Architecture

ERRORIST is composed of three modules: the `Error Generator`, the `Tracer`, and the `Evaluator` (Figure 2).

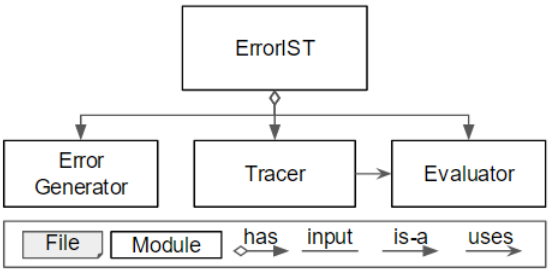

Figure 1: ERRORIST General Architecture

`Error Generator` generates the errors, `Tracer` analyse how the generated errors were corrected and Evaluator grades the modifications detected by the Tracer. In the following sections we describe these modules. A detailed presentation of the generated errors is presented in Section 3.

2.1 Error Generator

`Error Generator` receives as input a text, in which errors should be inserted. An error per sentence is inserted, being the error types specified by the user, who can also specify how many error of a certain type should be specified. If a certain error type, for some reason, cannot be applied to the current sentence, ERRORIST moves to the next sentence and reports the event.

`Error Generator` takes advantage of several resources, namely:

- a Part-of-Speech (POS) tagger for the appropriate language. The current version of ERRORIST uses the bigram tagger from NLTK (Bird et al., 2009) trained on Floresta Sintá(c)tica Corpus (Bick et al., 2007);
- a set of affix rules, which allows ERRORIST to move from a word form to another (for example, to transform nouns in verbs), move from one genre to the other, etc. As previously said, ERRORIST uses Ispell affix files, available for many languages.

In addition, ERRORIST also accepts as input files with handcrafted rules that capture some language particularities. For instance, rules that represent usual spelling errors in Portuguese.

Finally, `Error Generator` outputs a file with the generated errors.

2.2 Tracer

`Tracer` receives as input the `Correction File` (a file containing all the sentences modified by the editor) and uses the `Error Data` created by the `Error Generator` in order to identify changes in the sentence made by the editor. When an error is created, the relevant information is stored within a `Verification object`. What information is considered relevant varies from error type to error type, although most error types store the error placement, the original sequence, and the altered sequence. As an example, `Spelling Verification objects` contain the original word, the misspelled word and the word's placement in the sentence. The misspelling in "Bobby cried **wlof**!" would result in a `Verification object` storing "wolf", "wlof" and '2' as it is its position in the sentence (counting from 0). Due to their varying nature, each error type also has a specialized `Verification module`.

The `Tracer` uses the `Verification objects` stored in the `Error Data` in order to identify changes in the sentence. As of now, the `Verification objects` identify the changes by checking the modified sentence for three aspects:

Changed? Did the editor modify the inserted error?

Expected? Did the editor modify the inserted error to its correct, original form?

Other? Did the editor modify the sentence in a way that does not directly affect the error?

As an example, consider the sentence "Mary chased a lamb.", which was modified by ERRORIST into the following sentence, "ary chased a lamb." through a `Spelling error` in the word 'Mary'.

If the editor modifies it back to "Mary chased a lamb.", the error has been **Changed** and the modification was **Expected**. **No Other** modification has been made.

If the editor modifies it further to "Mary chased a little lamb.", the error has been **Changed** and the modification was **Expected**. **Other** modifications have been made, as the word 'little' was added.

If the editor modifies it to "Gary chased a lamb.", the error has been **Changed** and the modification was **Not Expected**. **No Other** modification has been made.

If the editor does not modify it, leaving the sentence as "ary chased a lamb.", the error was **Not Changed**. **No Other** modification has been made.

2.3 Evaluator

The `Evaluator`'s purpose is simple: to consider the modifications identified by the `Tracer` and to produce an adequate evaluation according to them. As such, its behavior can be summarized in the following input/output table (Table 1).

C?	E?	O?	Result
Yes	Yes	No	Ok
Yes	Yes	Yes	Undef
Yes	No	No	Undef
Yes	No	Yes	Undef
No	–	No	Ko
No	–	Yes	Undef

Table 1: `Evaluator` behavior

An **Ok** evaluation means that the correction does **not** need no further human verification as it is unquestionably **Correct**. A **Ko** evaluation means that the correction does **not** need further human verification as it is unquestionably **Incorrect**. An **Undef** evaluation means that the correction **needs** further human verification as the `Evaluator` could not determine if the correction was correct or incorrect.

For example, if the sentence "Mary chased a lmb." is corrected to "Mary chased a lamb.", it is classified as **Ok** as it was both **Changed** and **Expected**, having no **Other** modifications made to it. If it was corrected to "Mary chased a lion." It would be classified as **Undef** because, even though it was **Changed** and no **Other** modification was made to it, the change was not the **Expected** one. In another correction attempt, "Mary chased a little lamb.", the sentence is classified as **Undef** as well, because even though the error was **Changed** and the change was **Expected**, there were **Other** modifications made.

3 Generating errors

In the following section we resume the adopted error's taxonomy and, then, we describe how ERRORIST generates the different types of errors.

3.1 The error's taxonomy

ERRORIST implements the first three levels of the errors presented in the taxonomy detailed in (Costa et al., 2015) (Costa taxonomy from now on), namely: `orthography`, `lexis` and `grammar` errors (except the `untranslated` type). It also inserts `confusion of senses` errors from the `semantic` level. Here we summarize this taxonomy:

- Orthography
  - punctuation (ex: *I found **,** the clowns, Bob, and Clyde.*
  - capitalisation (ex: ***i** think my poor Slipper got dirty!*)
  - spelling (ex: *I have **htree** friends.*)
- Lexis
  - Omission (content or function words) (ex: *His hat was **[]***)
  - Addition (content or function words) (ex: *He bought a **already** hat.*)
- Grammar
  - Misselection
    - Word-class (ex: *The **cutely** bird is on the branch.*)
    - Verbs (tense, person or both) (ex: *He had **buy** a suit-case.*)
    - Agreement (gender, number, person or combinations)(ex: ***Os lobo** fugiu.*[3])
    - Contraction (ex: *Ela senta-se **em a** cadeira.*[4]).
  - Misordering (ex: *I **beautiful** like the [] color of your eyes.*)

[3] *The Wolf run away*. In Portuguese, *Os* is plural and *lobo* singular.

[4] *She seats in the chair*. In Portuguese, *em + o = em*

3.2 Ortography

Considering `orthographic errors`, `capitalization errors` are inserted by changing a word's initial letter (from capitalized to non-capitalized and vice-versa), and `punctuation errors` are created by simply replacing a punctuation mark with another punctuation mark. The latter method proved to be inadequate as they did not represent actual punctuation errors. For this reason, `Punctuation` errors were further refined into other generation methods. One method generates `Punctuation` errors by adding a comma or a semicolon between a subject and predicate or between a verb and its complements. To introduce the latter, ERRORIST uses the POS tagger, identifies nouns or pronouns (what we consider the subject) and then identifies a verb (the predicate) placing a comma before it. The same logic is applied when inserting a comma between the verb and its complements, where the verb is identified and if any adjective, noun, pronoun or article is found after it, a comma is placed after the verb. ERRORIST also inserts punctuation errors by removing a comma after an adverb. This is made possible by the POS tagger as well. ERRORIST first identifies an adverb and then checks the succeeding character for a comma. If found, ERRORIST removes it. Finally, `Spelling` errors are inserted by omitting, replacing, adding, or misplacing letters from a word. Considering that some errors are more plausible than others, either due to the involved language, or the used keyboard, ERRORIST also allows the user to add rules that will trigger specific spelling errors. This is possible through the `Sound Confusion File`. It should be provided by the user and each line should have two character sequences representing those errors. As an example, take these two pairs of character sequences:

`ãos` → `ões`

`ss` → `ç`

In Portuguese, the suffixes `ãos` and `ões` are used to indicate the plural forms of nouns terminated in `ão`. Thus, a possible error is obtained by replacing `ãos` by `ões`. Another possible (and common mistake, as both sequence sound the same) is when `ss` is replaced by `ç`.

3.3 Lexis

`Omission errors` are created by removing a word from the sentence; `Addition errors` are created by adding a word to the sentence. Both of these errors can be further specified, regarding whether a function word (e.g a conjunction) or a content word (e.g a noun or verb) should be omitted/added. To identify a word's class ERRORIST uses a POS tagger. The added words reside on a file provided by the user. This file should consist of several lines, each one containing a POS tag followed by words belonging to that tag. As an example:

```
  ART a the an
NOUN dog cat car log
VERB bark wait comb
```

Addition of content/function words depend on the word tags to distinguish them from the other word type.

3.4 Grammar

Excluding `Misordering` errors, every error type in the Grammar level is a `Misselection error`.

Most `Misselection` errors are inserted by altering the word's suffix according to rules provided by the user according to Ispell (Gorin et al., 1971) entries. These entries are comprised of a regular expression, a removal sequence, a substitution sequence and information regarding the suffix change (like word number or verb tense).

```
regex > -removal, substitution ;
"info_a=x, info_b=y"
```

ERRORIST also creates a reverse entry for each and every rule in order to allow suffix changing in both directions. For example, if the file only contains entries regarding suffix changes when turning a verb from the infinitive to each possible tense, changing the verb's tense would be impossible. Creating reverse entries allows ERRORIST to generate transition points that can be used to further alter the word. For example

```
I R > -R,AM # "P=3,N=p,T=pi"
```

It means that, to use this rule, the word must match "I R" at its end. Then, to alter it, the letter 'R' must be removed and replaced with the letters "AM". Using the rule to alter the word "dormir" (to sleep), which is possible because it

ends in "I R", we remove the letter 'R' from its end and replace it with "AM" which finally results in "dormiam" a past variant from the verb "dormir" (to sleep).

While these entries permit ERRORIST to alter verbs in tense and person, it does so using only the infinitive form, which is not ideal since most verb forms present in sentences are not in the infinitive form. For this reason, ERRORIST also creates a reverse entry for each of these rules in order to make sure each verb form can return to its infinitive form.

As an example the reverse entry for the entry above would be:

```
A M > -AM, I #
"P=3,N=p,T=pi,Reverse=yes"
```

Using the infinitive form as middle ground through these reverse entries, ERRORIST can change a verb tense or person through its infinitive form. Using this reverse entry to transform "dormiam" into "dormir", the verb's infinitive form, and then using another entry like:

```
I R > -R,STE # "P=2,N=s,T=pp"
```

We successfully transformed "dormiam" into "dormiste" (second person singular form of the past perfect) using the verb's infinitive form as a middle point.

As stated previously, most `Misselection` errors depend on affix files. These files have entries, like the ones above, that allow the modification of words through their suffixes. Despite this fact, not every modification is adequate for every word. It makes no sense to use an entry designed to transform adjectives into nouns if the word is already a noun for example. To cope with this situation, ERRORIST does not use a single affix file, but a set of affix files made out of the original, separated by purpose. ERRORIST uses a different file for: `Misselection: Verb`, `Misselection: Number`, `Misselection: Gender`, `Misselection: Word Class`.

This permits ERRORIST to apply the correct rules to the correct words. Each error that needs so, uses the POS tagger in order to distinguish which words should be considered candidates for error generation. For example, `Misselection: Verb Tense` chooses a random verb from the sentence and applies the entry as described above.

`Misselection: Word Class` works a bit differently, as its whole purpose is to be utilized on multiple word classes. By separating the affix files in categories and reading the categories' headers ERRORIST can use the appropriate rule to alter the candidate word's class (assuming it was correctly tagged). An example category looks like this:

```
flag *C:; "CAT=v,T=inf" #cao
T A R > -TAR,ÇÃO;
    "CAT=nc,G=f,N=s,FSEM=cao"
CIONAR > -IONAR,ÇÃO;
    "CAT=nc,G=f,N=s,FSEM=cao"
AIR > -IR,CÇÃO;
   "CAT=nc,G=f,N=s,FSEM=cao"
UIR > -IR,ÇÃO;
   "CAT=nc,G=f,N=s,FSEM=cao"
```

By interpreting the first line we conclude that this category's purpose is to alter words from verb to another class. Unfortunately, the file is static which means the tag here should be in accordance with the tags used by the POS tagger.

Categories do not exempt `Misselection: Word Class` errors from filtering the words in order to prevent erroneous modifications (e.g changing an article's class), even if they are unlikely to occur.

3.4.1 Misselection: verbs

`Verb Misselection errors` use the entries described in Section 3.4. During their insertion, ERRORIST uses the information provided in each entry (tense and/or person) to choose an appropriate entry to change the suffix.

Verb Tense errors make sure to change suffix using entries that maintain its person. Verb Person errors change suffix using entries that maintain its tense and, finally, Verb Blend errors change suffix using entries that keep neither tense, nor person.

An unfortunate limitation of using affix files to alter verbs, is its incapability to modify irregular verbs correctly. While there are entries regarding irregular verbs, ERRORIST has no way to distinguish them from regular verbs.

3.4.2 Misselection: word class

Word Class errors depend on the categories provided on the affix file, as described in Section **??**. Each category corresponds to a possible word tag. A word with a matching tag will be changed to another category using an entry from that category.

Some examples of a word class entry are:

```
flag *n:  ; "CAT=v,T=inf" #ente
I R > -IR,ENTE ;
"CAT=adj,N=s,FSEM=nte"
O R > -R,NENTE ;
"CAT=adj,N=s,FSEM=nte"
```

From the first line, this section comprises transformations applicable to verbs (from the "CAT" value). Each entry has its own category with the word class from the resulting word as its value. As an example, using the first entry we can turn the verb "aderir" (to adhere) into the adjective "adherent".

3.4.3 Misselection: agreement

Agreement Number errors are inserted by finding an appropriate entry to change the word's number, as described in Section **??**. Agreement Gender errors, due to the lack of entries related to gender switches in plural words, first turn the word to singular (if needed), then change the word's gender, and finally back to plural form. Using as an example, the word "pato" (duck) and the following entry:

```
[^Ã][^LSMRNZX]> S # "N=p"
```

By adding the letter 's', it would become "patos" (ducks). If there was the need for changing this word's gender, ERRORIST would first turn into singular by using the above entry's reverse entry. Then, using the following rule to change its gender:

```
[^Ã] O > -O,A # "G=f"
```

The word becomes "pata" (female duck). In order to maintain the word's original number, ERRORIST uses the first entry again, turning "pata" into "patas" (female ducks).

Agreement Person errors, unlike the above Misselection errors, does not rely upon affix files but instead relies on a Person Agreement File since the potential word changes are very few. The file should be provided by the user and each line like the following example:

```
meus teus seus nossos vossos
meu teu seu nosso vosso
```

Where each line has every variation possible for a possessive determiner.

3.4.4 Misselection: contractions

Contraction errors are inserted by finding a contracted word and separating it using a file composed of contractions and their respective composing parts. The file should be provided by the user and each line should consist of a contraction followed by its composing parts. As an example take these lines:

```
da de a
à a a
```

Using the first entry, ERRORIST is able to generate contraction errors by separating the word "da" into "de" and "a".

3.4.5 Misordering

Misordering errors are inserted by misplacing a word, randomly chosen, within the sentence. Two positions are randomly chosen, and the word in the first position is switched to the second. If, for some reason, the sentence remains unchanged (repeated consecutive words for example), the positions are picked again until the sentence is altered. As an example, take into account the sentence "I was cool." ERRORIST will pick a random word like "cool" and place it somewhere else as so: "**cool** I was []."

3.5 Semantic level

While the Costa taxonomy is aimed towards translation errors, most of its error types do not necessarily have a bad translation as their cause. As an example, the misspelling of the word "dog", "dawg", does not have to be the result of a mistranslation of the European Portuguese (EP) word "cão", it can happen by misspelling the word "dog". Conversely, a Confusion of Senses error can not happen outside of a translation context. As an example, using the word "cashier" instead of the "box" can only be considered as a `Confusion of Senses` error if you also consider its original wording in EP, "caixa". Until now, every error type discussed is inserted without taking into account a secondary language, but in Semantic level, this is not possible. With the exception of Wrong Choice errors, every other error in the Semantic level has a bad translation as its cause, which means we have to take into account the source language. To this end, we used Wiktionary (Wiktionary, 2016). Wiktionary is not a translation tool, but it serves the purpose of inserting these types of errors adequately. In `Confusion of Senses` errors, ERRORIST uses a language provided by the user in order to search for an adequate translation in the corresponding Wiktionary page. Then, it uses

the translated word's page to fetch all its meanings and returns one of the non-intended meanings. As an example, the English word 'chocolate' is translated to the French word 'chocolat' which, in turn, can be translated back to English as either 'chocolate', 'deceived' or 'tricked'. By replacing the original English word, 'chocolate', by either 'deceived' or 'tricked', we can successfully create a `Confusion of Senses` error.

4 Evaluation

In order to evaluate ERRORIST's quality, we must submit it to various tests. In the following sections we discuss the methods used for ERRORIST's testing, their reasoning, and, finally, their results. Every error in this Chapter was generated as described in The errors were inserted as described in Section 3. Each correction classification follows the rules described in Section 2.3.

4.1 Error Quality

ERRORIST inserts errors through insertion methods aiming to replicate error types found in the Costa taxonomy described in Section 3.1. Whether ERRORIST is able to perform such insertions correctly or not, needs to be ascertained.

As an example, a Misselection: Verb Tense error inserted in the verb "ter" (to have) could be inserted successfully by transforming it into "tinha" (had), which can be inserted by ERRORIST. Also inserted by ERRORIST is the transformation into "te", which, while a word in the EP language, is not a tense variation of "ter".

In order to evaluate ERRORIST's error insertion methods, we asked one of the original article writers, Ângela Costa to classify 6 errors of each error type found in ERRORIST (totalling 126) regarding whether or not they were according to the taxonomy described in Section 3.1.

As observed in Table 2, most errors accurately represent the Costa taxonomy error types at least 66.67% of the time (at least 4 out of 6 errors). Half of the `Word Class` and `Verb Tense` errors accurately represent their error typing. And `Verb Blend`, `Agreement: Gender`, `Agreement: Blend` and `Confusion of Senses` errors fail to represent their error typing accurately more than 33.33% of the time (2 out of 6). Surmising, 11 error types accurately represent their taxonomy in every error (100%), and only 4 error types fail to represent the taxonomy accurately in more than 50% of their examples.

	According	Not According
Punctuation	6	0
Punct Add	6	0
Punct Omit	6	0
Capitalization	6	0
Spelling	6	0
Omission F	6	0
Omission C	5	1
Addition F	6	0
Addition C	6	0
Word Class	3	3
Verb Tense	3	3
Verb Person	4	2
Verb Blend	1	5
Gender	1	5
Number	4	2
Person	6	0
Blend	2	4
Contraction	6	0
Misordering	6	0
Confusion	1	5
Total	84	42

Table 2: Error Quality

4.2 Traceability

In order to ascertain whether the error corrections can be detected automatically we used two different texts for error insertion.
The first text is a story written in EP (Story), and the second text is composed of translated film subtitles (Subtitles), each of the EP sentences was accompanied by the correct sentence in its native language, English. In order to provide varied errors for correction, we inserted 6 errors of each of the 21 types in the text, totaling 126 error prone sentences.

We extracted 126 sentences of each text to insert errors. Each of these sentences was automatically tagged using a bigram tagger from NLTK (Bird et al., 2009) trained on the Floresta Corpus (Bick et al., 2007). Each file had 126 error-prone sentences, each of the sentences was accompanied with a number identification and, in the Subtitle file, the original error-free sentence in English. We then assembled 10 persons and divided them randomly into to two groups of 5, one for each file. Every one of them is a native EP speaker and has a

good understanding of the English language. They were provided with instructions to correct the sentences in order for them to make sense. A breakdown for each error type in each file can be found in Table 4 and Table 5 for the Story and the Subtitles respectively.

	C?	E?	O?	Ok	Ko	T(%)
Story	454	411	163	334	110	70.48
Subtitle	489	386	210	325	73	63.17
Total	943	797	373	659	183	66.83

Table 3: Total scores

	C?	E?	O?	Ok	Ko	T(%)
Punctuation	29	25	5	23	0	76.67
Punct Add	14	14	5	12	13	83.33
Punct Omit	14	14	8	13	9	73.33
Capitalization	20	20	6	19	5	80.00
Spelling	26	24	9	17	2	63.33
Omission F	13	8	10	6	14	66.67
Omission C	15	8	10	8	9	56.67
Addition F	30	30	2	28	0	93.33
Addition C	25	25	4	23	3	86.67
Word Class	25	25	5	22	3	83.33
Verb Tense	27	24	2	23	2	83.33
Verb Person	28	27	7	21	2	76.67
Verb Blend	21	14	10	12	2	46.47
Gender	20	18	4	16	9	83.33
Number	14	14	7	10	13	76.67
Person	16	15	10	13	7	66.67
Blend	21	21	12	13	5	60.00
Contraction	22	21	14	11	5	53.33
Misordering	30	20	12	12	0	40.00
Confusion	14	14	14	9	7	53.33
No Error	30	30	7	23	-	76.67

Table 4: Story scores

As we can observe in table 3, only around 66.83% of corrections can be labelled as either correct or incorrect (**T(%)**) in a straightforward way.

4.3 Discussion

We can interpret **Changes** as the number of times the inserted error was actually noticed, since it was modified (correctly or not). In the subtitle file the number of **Changes** corrections is slightly higher, probably due to the added context the original sentence provides. Conversely, the number of

	C?	E?	O?	Ok	Ko	T(%)
Punctuation	26	16	8	15	4	63.33
Punct Add	23	21	6	19	5	80.00
Punct Omit	12	11	6	9	15	80.00
Capitalization	14	14	10	10	10	66.67
Spelling	27	22	9	20	0	66.67
Omission F	25	16	12	13	4	56.67
Omission C	25	19	11	19	0	63.33
Addition F	30	27	10	18	0	60.00
Addition C	30	22	13	17	0	56.67
Word Class	25	22	10	20	0	66.67
Verb Tense	27	19	9	18	1	63.33
Verb Person	19	5	12	4	6	33.33
Verb Blend	22	20	13	16	1	56.67
Gender	28	23	7	21	2	76.67
Number	19	17	15	14	1	50.00
Person	14	11	10	9	11	66.67
Blend	26	19	6	19	3	73.33
Contraction	19	16	13	12	4	53.33
Misordering	27	19	12	15	3	60.00
Confusion	21	17	12	13	3	53.33
No Error	30	30	6	24	-	80.00

Table 5: Subtitle scores

Expected corrections is lower for the same reason. Every translation presented was correct, still, the editors sometimes disagreed with the present translation and refined it to their terms, reducing the number of **Expected** corrections and raising the number of **Other** changes made in the sentences (as can be seen in Table 5).

The context, however, raised the number of both **Changed** and **Expected** significantly in `Omission` errors, as seen in Table 4 and 5, which indicates the error type is best used in non translation scenarios.

As we can observe in Table 3, the values on the tables for both files are very similar. The number of **Changed** and **Other** is slightly higher for the subtitles file. The traceability, the number of corrections ERRORIST can confidently determine as either correct or incorrect, for these errors (**T(%)**) is higher in the Story file, as well as the number of **Ko**. The number of **Expected** corrections for `Verb Person` errors in the Subtitle file is severely lower compared to the story file, again due to disagreements regarding the correct translation.

Regarding error quality, as expected, the errors belonging to more complex error types have

a lower reliability in their generation methods, as seen in Table 2. `Verb Tense`, `Verb Person` and `Verb Blend` all rely on `Affix files` for their generation methods. While `Affix files` are expressive regarding their suffix changes, they can also be used wrongly.

An unfortunate limitation of using this approach to alter verbs, is its incapability to modify irregular verbs correctly. While there are entries regarding irregular verbs, ERRORIST has no way to distinguish them from regular verbs.

As an example, the verb "ir" (to go), is an irregular verb but the following entry could still be applied:

```
I R > -R,STE # "P=2,N=s,T=pp"
```

This would result in the word "iste" which is not a word in EP.

Another probable cause is the ambiguity regarding which affix entry to use. If there is more than one entry fitting the suffix needed, ERRORIST chooses one them randomly, with the possibility of using a non ideal entry, resulting in less than adequate changes to regular verbs. `Agreement:Blend` errors also suffer from the same ambiguity regarding which affix entry to use.

`Confusion of Senses` errors rely on Wiktionary (Wiktionary, 2016) to generate them. The current implementation has 2 major issues. One being the randomness with which ERRORIST chooses the candidates for this error type's insertion. While Content words normally have different meanings to them, this is not the case with Function words, and they can be chosen just as easily. As an example, the EP article "o" is translated to "the". Translating it back to EP results in either "o", "os", "a" or "as", which are (excluding the original word), in fact, gender/number agreement errors of the same word.

5 Related Work

There are several taxonomies for MT errors, such as the ones described in (Bojar, 2011), (Vilar et al., 2006) and (Llitjós et al., 2005). According to (Costa et al., 2015), her taxonomy extends the previous ones, as it supports errors usually associated with morphological richer languages, such as Romance languages, which were not contemplated in those taxonomies. MQM (Multidimensional Quality Metrics) is a framework that, between others, targets translation quality metrics. Most of the considered error categories can be mapped into Costa's taxonomy, although some, mostly related to both design issues and encoding (which are out of the scope of ERRORIST) cannot. As Costa's taxonomy clusters the different types of errors in the main areas of linguistics (which allows for a precise understanding of the information level needed to generate the errors), we have adopted her taxonomy.

In what concerns error generation, several systems mention an error insertion step in order to replicate real error-prone corpora to train machine learning models on error detection and/or correction (for instance, (Foster, 2007), (Imamura et al., 2012), and (Felice and Yuan, 2014)). However, to the best of our knowledge, only two system focus specifically in this task: Missplel (Bigert et al., 2003) and the already mentioned GenERRate (Foster and Andersen, 2009).

Missplel (Bigert et al., 2003) is an automatic error insertion tool that although focused mainly on spelling errors, also inserts some Misselection errors. In Missplel errors are inserted them randomly.

In GenERRate (Foster and Andersen, 2009), considers four errors types: `Insertion`, `Omission`, `Move` and `Substitution`. The first three types can be directly linked to those in the Costa taxonomy (`Addition`, `Omission` and `Misordering`, respectively); `Substitution`, is a broad category that includes the remaining error types. Nevertheless, for the English language, GenERRate also provides some level of `Misselection` errors, namely `agreement` and `word class` errors, which are inserted by using handcrafted rules that change word number or turn a adjective into an adverb. Through similar rules, it can also insert verb Misselection errors by changing their tense. ERRORIST goes in deep into a fine-grained taxonomy, and also takes advantage of the aforementioned Ispell affix files.

6 Conclusions and Future Work

Even though artificial error insertion systems have been made in past, none of them were made with human evaluation as their main goal. For this purpose, we created configurable artificial error insertion tool, aiming to create errors adequate for editor/student evaluation. To accomplish this, ERRORIST is able to introduce errors according to

an extensive error taxonomy. ERRORIST is extensible by nature, on both error types and their verifiers.

While many error types proved to be generated adequately according to the Costa taxonomy, others proved to be unreliable. Some cases could be improved with some further refinement, like `Confusion of Senses` disregarding articles as candidates for error generation, while others would prove more difficult to improve, like `Misselection: Verb` errors due to requiring a more precise generation method both in verb type identification (regular vs irregular) and entry selection for suffix change.

If these amount of evaluations could lead to an accurate representation of the editor's quality or not, remains to be ascertained in the future. The lower number of **Expected** corrections in the Subtitle file could be explained by the presence of the original word in its native language, which can create disagreement regarding the translation. In order to verify this, further evaluation is needed but using the same source text for both the natural error-prone sentences and the error-prone sentences accompanied by the same sentence in its original language and the text accompanied.

Despite ERRORIST's shortcomings regarding error quality, it can both be used for automated error insertion and correction detection. It does not replace a human evaluator but it does reduce their work, since 66.83%, $(Ok + Ko)/Total$, of editor corrections can be detected automatically, even further (70.48%) if you consider non-translation editing.

ERRORIST and all the data gathered in this work will be made available upon publication.

References

Eckhard Bick, Diana Santos, Susana Afonso, and Raquel Marchi. 2007. Floresta sintá(c)tica: Ficção ou realidade? *Avaliação conjunta: um novo paradigma no processamento computacional da língua portuguesa. Lisboa, Portugal*, pages 291–300.

Johnny Bigert, Linus Ericson, and Antoine Solis. 2003. AutoEval and Missplel: two generic tools for automatic evaluation. *NODALIDA 2003 - Nordic Conference of Computational Linguistics*, 3.

Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

O. Bojar. 2011. Analysing Error Types in English-Czech Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, pages 63–76.

Chris Brockett, William B Dolan, and Michael Gamon. 2006. Correcting esl errors using phrasal smt techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 249–256. Association for Computational Linguistics.

Ângela Costa, Wang Ling, Tiago Luís, Rui Correia, and Luísa Coheur. 2015. A linguistically motivated taxonomy for machine translation error analysis. *Machine Translation*, 29(2):127–161.

Mariano Felice and Zheng Yuan. 2014. Generating artificial errors for grammatical error correction. In *EACL*, pages 116–126.

Jennifer Foster and Øistein E Andersen. 2009. Generrate: generating errors for use in grammatical error detection. In *Proceedings of the fourth workshop on innovative use of nlp for building educational applications*, pages 82–90. Association for Computational Linguistics.

Jennifer Foster. 2007. Treebanks gone bad: Parser evaluation and retraining using a treebank of ungrammatical sentences. *International Journal on Document Analysis and Recognition*, 10(3-4):129–145.

RE Gorin, Pace Willisson, Walt Buehring, Geoff Kuenning, et al. 1971. Ispell, a free software package for spell checking files. *The UNIX community*.

Kenji Imamura, Kuniko Saito, Kugatsu Sadamitsu, and Hitoshi Nishikawa. 2012. Grammar error correction using pseudo-error sentences and domain adaptation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 388–392. Association for Computational Linguistics.

Ariadna Font Llitjós, Jaime G Carbonell, and Alon Lavie. 2005. A Framework for Interactive and Automatic Refinement of Transfer-Based Machine Translation. In *In Proceedings of EAMT 10th Annual Conference*, pages 30–31.

David Vilar, Jia Xu, Luis Fernando D'Haro, and Hermann Ney. 2006. Error Analysis of Machine Translation Output. In *International Conference on Language Resources and Evaluation*, pages 697–702, Genoa, Italy, May.

Wiktionary. 2016. Wiktionary, a collaborative project to produce a free-content multilingual dictionary. Accessed 30-09-2016.