RACAI GEC – A hybrid approach to Grammatical Error Correction

Tiberiu Boroș
Calea 13 Septembrie, 13
Bucharest
RACAI
tibi@racai.ro

Stefan Daniel Dumitrescu
Calea 13 Septembrie,13
Bucharest
RACAI
sdumitrescu@racai.ro

Adrian Zafiu
Str. Targul din Vale, 1
Pitesti
UPIT - FECC
adrian.zafiu@comin.ro

Dan Tufiș
Calea 13 Septembrie, 13
Bucharest
RACAI
tufis@racai.ro

Verginica Mititelu Barbu
Calea 13 Septembrie, 13
Bucharest
RACAI
vergi@racai.ro

Paul Ionuț Văduva
Calea 13 Septembrie, 13
Bucharest
RACAI
ionut@racai.ro

Abstract

This paper describes RACAI's (Research Institute for Artificial Intelligence) hybrid grammatical error correction system. This system was validated during the participation into the CONLL'14 Shared Task on Grammatical Error Correction. We offer an analysis of the types of errors detected and corrected by our system, we present the necessary steps to reproduce our experiment and also the results we obtained.

1 Introduction

Grammatical error correction (GEC) is a complex task mainly because of the natural dependencies between the words of a sentence both at the lexical and the semantic levels, leave it aside the morphologic and syntactic levels, an intrinsic and complex attribute specific to the human language. Grammatical error detection and correction received a significant level of interest from various research groups both from the academic and commercial environments. A testament to the importance of this task is the long history of challenges (e.g. Microsoft Speller Challenge and CONLL Shared Task) (Hwee et al., 2014) that had the primary objective of proving a common testing ground (i.e. resources, tools and gold standards) in order to assess the performance of various methods and tools for GEC, when applied to identical input data.

In the task of GEC, one can easily distinguish two separate tasks: grammatical error detection and grammatical error correction. Typically, there are three types of approaches: statistical, rule-based and hybrid. The difficulty of detecting and correcting an error depends on its class.

(a) Statistical approaches rely on building statistical models (using surface forms or syntactic labels) that are used for detecting and correcting local errors. The typical statistical approach is to model how likely the occurrence of an event is, given a history of preceding events. Thus, statistical approaches easily adaptable to any language (requiring only training data in the form of raw or syntactically labeled text) are very good guessers when it comes to detecting and correcting collocations, idioms, typos and small grammatical inadvertences such as the local gender and case agreements. The main impediments of such systems are two-fold: (1) they are resource consuming techniques (memory/storage) and they are highly dependent on data – large and domain adapted datasets are required in order to avoid the data-scarceness specific issue and currently they rely only on a limited horizon of events; (2) they usually lack semantic information and favoring high-occurring events is not always the best way of detecting and correcting grammatical errors.

(b) Rule-based approaches embed linguistic knowledge in the form of machine parsable rules that are used to detect errors and

Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task, pages 43–48,
Baltimore, Maryland, 26-27 July 2014.

describe via transformations how various error types should be corrected. The drawbacks to rule-based system are (1) the extensive effort required to build the rule-set, (2) regardless of the size of the rule-set, given the variability of the human language it is virtually impossible to capture all possible errors and (3) the large number of exceptions to rules.

(c) Hybrid systems that combine both rule-based and statistical approaches are plausible to overcome the weaknesses of the two methodologies if the mixture of the two components is done properly. Detection of errors can be achieved statistically and rule-based, the task of the hybrid approach being to resolve any conflicts that arise between the outputs of the two approaches.

However, even the most advanced systems are only able to distinguish between a limited number of error types and the task of correcting an error is even more difficult. Along the typical set of errors that are handled by typical correction systems (punctuation, capitalization, spelling, typos, verb tense, missing verb, etc.), CONLL's GEC task introduces some hard cases which require a level of semantic analysis: local redundancy, unclear meaning, parallelism, etc.

2 External tools and resources

One step in the preparation phase was the analysis of the types of errors. The training set was automatically processed with our Bermuda tool (Boroș et al., 2013): it underwent sentence splitting, tokenization, part of speech tagging, lemmatization and also chunking. Comparing the original and the corrected sentences, we could rank the types of mistakes.

The most frequent ones, i.e. occurring more than 1000 times, are presented in the following table:

Type of error	Occurrences	Percent
use of articles and determiners	6647	14.98
wrong collocations or idioms	5300	11.94
local redundancies	4668	10.52
noun number	3770	8.49
tenses	3200	7.21
punctuation and orthography	3054	6.88
use of prepositions	2412	5.43
word form	2160	4.87
subject-verb agreement	1527	3.44
Verb form	1444	3.25
Link word/phrases	1349	3.04

Table 1. The most frequent types of mistakes in the training data

There are also some less frequent errors: pronoun form, noun possessive form, word order of adjectives and adverbs, etc. Some of these can be solved by means of rules, others by accessing lexical resources and others are extremely difficult to deal with.

As far as the test data are concerned, the error distribution according to their types is the following:

Type of error	Occurrences in official-2014.0.m2	Occurrences in official-2014.1.m2
use of articles and determiners	332	437
wrong collocations or idioms	339	462
local redundancies	94	194
noun number	214	222
tenses	133	146
punctuation and orthography	227	474
use of prepositions	95	152
word form	76	104
subject-verb agreement	107	148
Verb form	132	88
Link word/phrases	93	78

Table 2. The most frequent types of mistakes in the test data

Roughly, the same types of mistakes are more frequent in the test set, just like as in the training set.

For collocations and idioms, as well as for correct prepositions use, we consider that only lexical resources can be of help. They can take the form of a corpus or lists of words that subcategorize for prepositional phrases obligatorily headed by a certain preposition (see section 3.2). We adopted the former solution: we used Google 1T n-grams corpus (see section 2.2) from which the selectional restrictions can be learned quite successfully. However, dealing with collocations is difficult, as correction does not involve only syntax, but also semantics. Changing a word in a sentence usually implies changing the meaning of the sentence as a whole. Nevertheless, a solution can be found: as mistakes in collocations involve the use of a related word (synonyms), a

resource such as the WordNet can be of help. When the word used in the sentence and the one occurring in the corpus can be found in the same synset (or even in synsets in direct relation), the correction could be made. Otherwise, it is risky to try. In any scenario, this remains as future work for us.

2.1 RACAI NLP Tools

We have used our in-house Bermuda software suite (Boroș et al., 2013), (Boroș and Dumitrescu, 2013) to perform text preprocessing. As the tool is well documented in the cited papers above, we summarize its main functionalities concerning the task at hand and the algorithms behind them.

Tokenization. A basic necessary preprocessing step that needs to be applied from the beginning as most tools work on a certain tokenization format. Bermuda uses a custom-built, language dependent tokenizer. Based on regular expressions, it detects and splits words such as [haven't] into [have] and [n't]; [boy's] into [boy] and ['s], while leaving abbreviations like [dr.] or [N.Y.] as a single token.

Part-of-speech (POS) tagger. Tagging is essential to determine each word's part of speech and thus its role in the sentence. Each word is tagged with a morpho-syntactic descriptor, called MSD. The English language has around 100 MSDs defined, while more inflected languages, like Romanian – a Latin-derived language, uses over 600. An MSD completely characterizes the word morphologically and syntactically [1]. For example, 'Np' refers to a proper noun while 'Ncns' refers to a common (c) noun (N) that has a neuter (n) gender and is in singular form (ex: zoo, zone). Our tagger is based on a neural network, introduced in (Boroș et al., 2013). Overall, the Bermuda POS Tagger obtains very high accuracy rates (>98%) even on the more difficult, highly inflected languages.

Lemmatization. The Bermuda Lemmatizer is based on the MIRA algorithm (Margin Infused Relaxed Algorithm) (Crammer and Singer, 2003). We treat lemmatization as a tagging task, in which each individual letter of the surface word is tagged as either remaining unchanged, being removed or transformed to another letter. The lemmatizer was trained and tested on an English lexicon containing a number of around 120K surface-lemma-MSD entries.

[1] Full description of MSDs can be found at : http://nl.ijs.si/ME/V4/msd/html/msd-en.html

2.2 Google 1T corpus

A good performing language model is a very important resource for the current task, as it allows discriminating between similar phrases by comparing their perplexities.

Although we had several corpora available to extract surface-based language models from, we preferred to use a significantly larger model than we could create: Google 1T n-gram corpus (Brants and Franz, 2006). This 4 billion n-gram corpus should provide high-quality perplexity estimations. However, loading 4*10^9 n-grams without any compression scheme would require, even by today's standards, a large amount of memory. For example, using SRILM (Stolcke, 2002) which uses 33 bytes per n-gram, would require a total of ~116GB of RAM. The article by Adam Pauls and Dan Klein (2011) describes an ingenious way to create a data structure that reduces the amount of RAM needed to load the 1T corpus. However, the system they propose is written in Java, a language that is object-oriented, and which, for any object, introduces an additional overhead. Furthermore, they do not implement any smoothing method for the 1T corpus, defaulting to the +1 "stupid smoothing" as they themselves named it, relying on the fact that smoothing is less relevant with a very large corpus. For these reasons, coupled with the difficulty to understand and modify other persons' code, we wrote our language model software. We based our implementation around Pauls and Klein's sorted array idea, with a few modifications. Firstly, we encoded the unigrams in a simple HashMap instead of a value-rank array. Secondly, we wrote a multi-step n-gram reader and loader. Thirdly, we implemented the Jelinek-Mercer smoothing method instead of the simple +1 smoothing. Using deleted interpolation we computed the lambda parameters for the JM smoothing; we further built a stand-alone server that would load the smoothed n-gram probabilities and could be queried over TCP-IP either for an n-gram (max 5-gram – direct probability) or for an entire sentence (compute its perplexity). The entire software was written in C++ to avoid Java's overhead problems. Overall, the simplified ranked array encoding allowed us to obtain very fast response times (under a millisecond per query) with a moderate memory usage: the entire 1T corpus was loaded in around 60GB of RAM, well below our development server memory limit.

We are aware of the limitations of this corpus: as data was collected from the web, mistakes will occur in it.

We also need a language model that can estimate the probability of parts of speech. By learning a model from the parts of speech we can learn to discriminate between words different forms. Grantedly, a part of speech language model can promote a grammatically "more" correct but semantically inferior sentence over a semantically sound one, due to assigning a higher probability to a more common part of speech sequence in the sentence. Our experiments show that, generally, a part of speech language model helps text quality overall.

Our initial idea for this POS language model was to use the same 1T corpus that we could annotate using our tagging tools. However, given the limited context, performance would have been acceptable at the 5-gram level, decreasing to the point of simply picking the most common part of speech for the unigrams, as no context exists for them. As such, we used the following available monolingual resources for English: the News CRAWL corpus (2007-2012 editions), Europarl, UN French-English Corpus, the News Commentary, our own cleaned English Wikipedia dump. The total size of the raw text was around 20GB. We joined and annotated the files and extracted all the 1-5 grams, using the same format as the 1T corpus. We then used another instance of the language model software to load this POS LM and await the main system perplexity estimation requests. Overall, the part of speech language model turned out to be rather small (a hard-disk footprint of only 315MB of binary part of speech LM compared to the 57GB of surface model compressed data). This is normal, as the entire part of speech MSD vocabulary for English is around 100 tags, compared to the more than 13 million surface forms (unigrams) in the 1T corpus.

3 RACAI's Hybrid Grammatical Error Correction System

3.1 An overview of the system

In many cases, statistical methods are preferable over rule-based systems since they only rely on large available raw corpora instead of handcrafted rules that are difficult to design and are limited by the effort invested by human experts in their endeavor.

However, a purely statistical method is not always able to validate rarely used expressions and always favors frequency over fine grained compositions.

As a rule of thumb, hybrid systems are always a good choice in tasks where the complexity exceeds the capacity of converting knowledge into formal rules and large scale training data is available for developing statistical models.

Our GEC system has three cascaded phases divided between two modules: (a) in the first phase, a statistical surface based and a POS LM are used to solve orthographic errors inside the input sentences, thus enhancing the quality of the NLP processing for the second stage; (b) a rule-based system is used to detect typical grammatical errors, which are labeled and then (c) corrected using a statistical method to validate between automatically generated candidates.

3.2 The statistical component

Typos are a distinctive class of errors found in texts written by both native and non-native English speakers which do not violate any explicit (local agreement related) grammatical constraints. Most POS tagging systems handle previously unseen words through suffix analysis and are able (using the local context) to assign a tag which is conformant with the tags of the surrounding words. Such errors cannot be detected by applying rules, since it is impossible to have lexicons that cover the entire possible vocabulary of a language.

The typical approach is to generate spelling alternatives for words that are outside the vocabulary and to use a LM to determine the most likely correct word form. However, when relying on simple distance functions such as the unmodified Levenstein it is extremely difficult to differentiate between spelling alternatives even with the help of contextual information. There are multiple causes for this type of errors, starting from the lack of language knowledge (typically non-native speakers rely on phonetic similarity when spelling words) to speed (usually results in missing letters) or keyboard related (multiple keys touched at once). The distance function we used for scoring alternatives uses a weighted Levenstein algorithm, which was tuned on the TREC dataset.

3.3 The rule based error detection and correction

As previously mentioned, not all grammatical errors are automatically detectable by pure statistical methods. In our experiments we noticed frequent cases where the LM does not provide

sufficient support to distinguish between true grammatical errors and simply unusual but grammatically correct expressions.

For the present shared task we concentrated on a subset of potential errors. Our rules aimed the correction of the verb tense especially in time clauses, the use of the short infinitive after modals, the position of frequency adverbs in a sentence, subject-verb agreement, word order in interrogative sentences, punctuation accompanying certain lexical elements, the use of articles, of correlatives, etc.

For the sake of an easier understanding of our rule-based component of the GEC system, we will start by introducing some technical details about how the rule interpreter works, emphasizing on the structure of the configuration file, the input modality and the general pointers on writing rules. In our approach we treat error detection and error correction separately, in a two-stage system. The configuration file contains a set of language dependent rules, each rule being uniquely identified by the label and its body. The role of using labels is two-fold: (1) they provide guidance and assistance to the user in navigating through the structure of the configuration file (when editing or creating new rules); (2) they play a crucial role in the error correction process and serve as common denominators for different classes of errors.

Our rule description system is inspired after the time-independent logic function (combinational logic) paradigm, which stipulates that a fixed input size logical function, described through a stochastic list of input/output dependence sequence, through a process of logical minimization, this function can be implemented as an array of "AND" gates, followed by an array of "OR" gates. Thus, in our configuration file, each rule is described by a set of string pairs (i_0 r_0, i_1 r_1… i_n r_n) which act as "AND" gates – we refer to this as a sub-instance of a rule. At this point, a sub-instance is activated only if all constraints are met. The "OR" gate array is simulated by adding rules with the same label. This way, if any sub-instance is active then the rule is considered active and we proceed to the errror correction step.

Every pair (i_k r_k) is a single Boolean input of a sub-instance. A rule is checked against every token inside an utterance, from left to right. r_k is a regular expression which, depending on the value of i_k, is applied to the word's surface form (s), the word's lemma (l) or the word's MSD (m). i_k can also select if the regular expression should be applied to a neighboring token. To exemplify, we have extracted two sections from our configuration file: (a) the modal infinitive common error for non-native English speakers (also found in the development set of CONLL) (lines 1 to 7) and (b) the possible missing comma case (line 8):

```
1) modal_infinitive: s must   s+1 to s-1 ^!a
2) modal_infinitive: s could  s+1 to
3) modal_infinitive: s can    s+1 to
4) modal_infinitive: s might  s+1 to
5) modal_infinitive: s may    s+1 to
6) modal_infinitive: s would  s+1 to
7) modal_infinitive: s should s+1 to
8) pmc: s which m-1 ^((?!COMMA).)*$
```

Table 3: a sample of error detection rules

The "modal_infinitive" rule is complex and it is described using 7 sub-instances, which share an identical label. Line 1 of the configuration excerpt contains three pairs as opposed to the other sub-instances. This does not contradict the combinational logic paradigm, since we can consider this rule as having a fixed input size of three and, as a result of logic minimization, the third parameter for 6 of the seven instances falls into the "DON'T CARE" special input class. The first $i_k r_k$ pair ("s must") is used to check if the surface form ("s") of the current word is "must". The second pair ("s+1 to") checks if the word form of the next token is "to". The third pair ("s-1 ^!a") verifies that the collocation "a must to" does not accidentally trigger this rule. This rule will detect the error in "I must to go...", but will licence a sequence like "This book is a must to read...".

The error detection rules that we designed for the CONLL shared task are created, as an external resource for the program, on the basis of the mistakes observed in the training set and can be updated/extended any time .

In the error correction phase, for every error type we encompass, we provide the necessary transformations (at token level) through which the initial word sequence that generated this error should be corrected. The configuration file of this module is straightforward: rule-labels are marked as strings at the beginning of a new line; for each label, we provide a set of transformation rules, that are contained in the following tab-indented lines; once a new line does not start with a TAB character, it should either be empty or contain the label for a different error type. the correction phase, multiple sentence candidates are automatically generated (based on the transformation rules) and they are checked against the

language model to see which one yields the lowest perplexity. That is, once an error is found, its correction way tends to be applied provided that the language model offers another solution.

As an example, suppose that the rule for detecting a possible missing comma (pmc in Table 3, line 8) was fired. The corresponding correction rule is described as below:

```
pmc:
   $w-1 , $w
   $w-1 $w
```

The "pmc" rule is activated if the word "which" is not preceded by a comma. Since it is not always the case that the wordform "which" should be preceded by this punctuation mark, in our error correction system step we generate two candidates: (a) one in which we insert a comma before "which" and (b) one in which we keep the word sequence untouched.

4 Results and Conclusions

The RACAI hybrid GEC system obtained a precision of 31.31%, a recall of 14.23% and an $F_{0.5}$ score of 25.25% on the test set provided by the CONLL shared task on Grammatical Error Correction.

We presented our system and the resources we used in the development process. All the data and tools required to run a similar experiment are available online and we are currently working on developing a self-contained GEC system that will be made publicly available.

Future development plans include the enhancement of the lexicons we use for English and the extension of this system for Romanian. Furthermore, we plan to include an extended method for solving collocations errors based on the synsets of Princeton WordNet (PWN) (Fellbaum, 1989).

References

Andreas Stolcke. 2002. SRILM: An extensible language modeling toolkit. In Proceedings of Interspeech

Boroş, T., Radu, I., & Tufiş, D. (2013). Large tagset labeling with Feed Forward Neural Networks. Case study on Romanian Language. In Proceedings of ACL

Boroş, T., & Dumitrescu, S. D. (2013). Improving the RACAI Neural Network MSD Tagger. In Engineering Applications of Neural Networks (pp. 42-51). Springer Berlin Heidelberg

Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. The Journal of Machine Learning Research, 3, 951-991.

Fellbaum, Ch. (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant (2014). The CoNLL-2014 Shared Task on Grammatical Error Correction. Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task (CoNLL-2014 Shared Task). Baltimore, Maryland, USA.

Thorsten Brants and Alex Franz. 2006. Google Web1T 5-gram corpus, version 1. In Linguistic Data Consortium, Philadelphia, Catalog Number LDC2006T13

Pauls, Adam, and Dan Klein. "Faster and smaller n-gram language models." Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 2011.