Grammar Checking using POS Tagging and Rules Matching

Zac Rider
Computer Science Department
Swarthmore College
Swarthmore, PA 19081
rider@cs.swarthmore.edu

Abstract

This paper is an examination of various techniques that could be used for grammar checking and the description of the results that were generated using a simple rules matching system. To generate the rules for this system, two techniques were considered: hand construction and an algorithm that randomly generates large numbers of rules and uses comparison against large corpora to find valid rules. While individual construction of rules proved to be effective for addressing specific errors, the random algorithm proved to be effective for a larger number of grammatical errors.

1 Introduction

There's something wrong with the sentence: *Microsoft company should big improve Word grammar check*, but Word 2004 thinks that the only problem is that *company* should be capitalized. Grammar checking is one of the more complicated tasks for word processing, and the more irregular and exception-filled the language, the more difficult the problem becomes. Problems such as a noun-verb mismatch: *one of the mistakes are bad*, or adjectives incorrectly used as adverbs: *I can't read so good*, are much easier to find than a somewhat ambiguous mistake such as: *The badger was acted upon* (passive voice).

The simplest method of fixing grammatical errors, which was used for the experiments for this project, is the process of rules matching, that is, constructing a rule that applies to a given grammar and then checking that the given input follows, or does not follow, that rule. Using lexigraphically aided finite state machines is another, more complicated method, that combines a bootstrapped learning algorithm with parsing and POS tagging (Sofkova Hashemi et al., 2003). Other methods include syntactic analysis and parse tree analysis (Bender et al., 2004).

One thing that differs in the methods of grammar checking systems is whether or not the system is checking for negative or positive grammar. Intuitively, it seems like it might be easier to define the properties that are correct in a grammar, as there are a set number of grammatical configurations that are correct and an infinite number of configurations that are incorrect. The problem is that describing all of the correct configurations for a grammar checker requires that for every check, it must look at every single rule to see if a given example is in the grammar. This process is necessarily slower than a system that uses a relatively few rules per check to see if something is *not* in the grammar. Since speed is not of great concern for the system in this paper, the rules checking could have been implemented either way, but for simplicity, we chose to implement rules that check for specific errors in grammar instead of using a model of correct grammar to find incorrect examples. For a small system, it is easier to describe a few things in English that are grammatically incorrect than every rule that is correct.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

2 Related Work

One approach to checking grammar relies on a technique called *aligned generation* (Bender et al., 2004). However, this process is not used in the everyday sort of grammar checking that might be used in a word processor, rather it is a complicated process that takes a fair amount of time and is used for generating language learning systems. The system takes *mal-rules* and *mal-lexical types and entries* given by the user and uses *feature structure grammar* analysis, which is an extensive search of multiple parse trees for errors based on the given rules. The majority of the work in the system is the parsing itself, in which the input sentence is put into every possible configuration, and then those configurations are rated, and an acceptable configuration is chosen. One concern with this method is that the process of creating the parse trees for analysis is potentially time consuming.

Finite state machine analysis has the interesting property of not being a rules based system, rather it is a bootstrapped learning system that uses regular expressions along with FSMs to attempt to judge the correctness of lexically determined phrases (Sofkova Hashemi et al., 2003). The phrases generated by the system's lexicon are strings mapped to a tag containing part-of-speech and other feature information. While this method has 92% recall, it only has approximately 45% precision. This could prove cumbersome for a word processor system, as the user could be presented with many cases that the checker flags as errors that are, in fact, correct. However, for the task described in this paper, the recall percentage is acceptable. The random rules generator described in this paper is an approximation of this type of analysis, but the system detailed in this paper has no lexigraphical aids.

The system that this paper attempts to emulate is the Granska rules matching system (Domeij et al., 1999), which makes a point of not using Hidden Markov models and simply using what the author calls *error rules* to locate errors and *helping rules* to attempt to determine the best correction, and thus the best fitting rule for a given error. The Granska system has precision and recall of approximately 80% for the problems that it was designed for, namely noun-phrase disagreement and incorrectly split compounds in Swedish. The issue with the Granska system, however, is that while it has good results for these two problems, it turns out that the methods used in Granska do not translate well to all problems in grammar.

3 Parts of a Grammar Checker

A typical grammar checker that might be found in a word processor consists of three different pieces. First, a processor has to be able to separate the input into individual sentences. Then, it needs a part-of-speech (POS) tagger that can accurately label the data that it has. Charniak has an excellent analysis of POS tagging (Charniak et al., 1993) that is used by the makers of the Granska system. The particular POS tagger that is used for this system was taken from the Stanford website `http://www-nlp.stanford.edu/links/statnlp.html` (Toutanova and Manning, 2000; Toutanova et al., 2003) and works in log linear time. The speed of this system substantially speeds up training and test, as tagging is a necessary preprocessing step.

One issue that is of some concern for this system is that of POS tagger granularity. Some of the grammatical errors in English are fairly fine grained (ie. *was* vs. *were*), and because a POS tagger may not differentiate between the two, it makes it very difficult to attempt to detect problems associated with them. From a tagging perspective, the sentence *I wish I was dead* is the same as *I wish I were dead*, while from the perspective of a grammar checker, the second is correct and the first is not. While this particular example is not difficult to correct, it is a recurring problem that highlights the fact that when hand-constructing error rules it is easy to for them to become over-trained. When the granularity of the POS tagger isn't fine enough, a grammar checker which relies solely on POS tags will not be able to distinguish between many pairs of grammatical and ungrammatical sentences such as the ones illustrated above.

Finally, the system needs a method of identifying grammatical errors. In the case of Granska (Domeij et al., 1999), they exclusively use error rules matching. Rules matching has the convenient properties of being fast, easy to implement, and accurate for

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

the set of problems that the rules are constructed for. The regular expression analyzer and aligned generation systems are more suitable for larger scale systems that attempt to evaluate grammars as a whole.

4 Procedure

4.1 Rule Construction by Hand

Taking heavily from the ideas of the Granska system, the grammar checker created for this project essentially searches for a set of grammatical conditions and then flags something as an error if those conditions are found. For example, for a noun-verb mismatch the checker searches for a noun and then a verb. If the noun is singular and the verb is plural or vice versa, the phrase is noted as incorrect and the rules that are violated are recorded. What makes the process of rule constructing difficult, is that no rule is ever without exception. In addition to looking for a noun and a verb, the checker must also be able to ignore any possible prepositional phrase in between.

The rules system takes a given sentence and then runs every single rule in sequence. Rules can be added or subtracted depending on which grammatical error the user is looking for. Essentially, every rule is a small finite state machine. Rather than using actual words, the rules only check the POS tags of words. The size of a given rule is the number of POS tags that the rule contains. For each sentence, the grammar checker invokes each rule, which then checks itself against the sentence. This method has been implemented as a depth-first search of the sentence. First, the rule checks to see whether the tag of the current word in the sentence matches the first tag in the rule. If it does, the checker cycles to the next word to see if its tag matches the next tag of the rule, and so on for the whole sentence. In the case of a wild card tag, the checker simply cycles until it detects that the tag it is considering is the next tag in the sequence of the rule. If the next tag is never found, then the machine simply returns false.

For example, one of the specific rules for noun-verb mismatch contains the POS tags: {NNS, PP, *, NN, VBZ}. This rule, containing 5 POS tags, is size 5, and the '*' symbol stands for a wild card. For the sentence *The dogs of war is released*, the rule identifies *dogs* as the noun, then the preposition, which is *of*. The next noun is the object of the preposition, *war*, but there may be any number of POS tags in between *of* and *war*, because of the wild card tag. After *war*, the verb, *is*, is associated with the noun *dogs*, which is grammatically incorrect. Since the POS tags in the sentence follow the sequence in the rule, the sentence is flagged.

In the case of the sentence, *The dogs is eating the food*, a different rule is needed to catch the mistake. A rule containing the sequence: {NNS, VBZ} would work in theory, but then the sentence *The problem with the dogs is that they are bad.* would also be flagged as incorrect even though it is not. Examples such as these necessitate different levels of rules. This system has three different classifications of rules: specific, general, and improbable. Specific rules, such as: {NNS, PP, *, NN, VBZ} have the longest definitions. Specific rules have the highest probability of finding actual errors and not mistaking good sentences for bad sentences. General rules typically are just a little simpler than specific rules. If a specific rule would examine five tags, a general rule would examine two tags with a wildcard like: {NNS, *, VBZ}. Improbable rules are rules that more often than not are actually grammatically correct, but could be incorrect, like the example: {NNS, VBZ}.

For some problems such as noun-verb disagreement, it's a simple matter to figure out that the system should be looking for a singular noun followed by a plural verb, or vice versa, but for something like the *they're, there, their* problem, it's more complicated. Some rules used for this system can be found at EduFind Online: `http://www.edufind.com/english/grammar/`, but require a subscription to use. While the definitions on EduFind Online are more like a grammar primer than a programmer's guide to grammar checking, the rules that it has are fairly comprehensive and can easily be converted to POS tag following rules. For example, the description given for nouns, in which the site lists rules for each different form of noun. The rules include which forms of verbs are correct which forms of nouns, as well as exceptions to each rule and example sentences for each rule.

This kind of rules matching for the English language can become very complicated, and for trickier grammatical errors, the process of defining specific rules can be very difficult. Trying to process higher

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

level/difficulty errors requires the test cases to be so specific that the entire point of having generalized rules is lost.

4.2 Random Rule Construction

To try to extend the kind of rules matching in Granska to a larger scale, the other method attempted for determining rules was random rules generation. Writing upwards of 100,000 rules by hand is a daunting process, so the system randomly assigns POS tags to rules of a user defined size. One issue with the generation process is that it could create a rule of size 5 such as {VB, VBG, VBP, VB, VBG}. While this rule is trivially incorrect, the fact remains that it is incorrect. Therefore, while this generation system does create rules that don't exist, it is always possible that a person will write a pattern that should not exist that must be marked as incorrect. However, the generator might make a rule such as: {NN, PP, *, NN, VB}, which is grammatically correct.

In order to remove all of the rules that reflect correct grammar, the system tests the randomly generated rules against a corpus of correct English and then eliminates all of the rules that generate flags in the corpus, thus leaving a set of tags that hopefully do not reflect proper grammar. Using sheer numbers, this method attempts to keep all of the rules that reflect whatever English isn't. This method takes away the issue of having to write out rules by hand at the expense of rule precision and transparency. For this method, each rule is weighted equally, and there are no specific, general, or improbable rule designations. This particular method also has a fairly long training process.

5 Results

One of the difficulties of the hand-constructed rules was actually measuring the effectiveness of the result. For a rule like noun-verb mismatch, it is very difficult to actually be able to tell how well the system can find errors, because it is easier to find corpora that are correct than corpora that intentionally make mistakes and then make note of those mistakes. The system was tested on 50 manually generated sentences that contained noun-verb mismatches followed by 50 sentences that were grammatically correct.

The specific rules for noun-verb mismatches flagged 22/50 of the incorrect sentences and 1/50 of the correct sentences as incorrect. General rules flagged 37/50 the incorrect sentences and 26/50 of the correct sentences. The improbable rules flagged 48/50 of the incorrect sentences and 39/50 of the correct sentences as incorrect. The precision and recall of the different rule types are shown in Figure 1.

The system was also tested on the *their/there/they're* and the *then/than* problem. Although these problems are actually contextual spelling errors, they can still be found with this system. The tests using these problems generated results similar to the noun-verb mismatch problem. The major issue with dealing with results for these specific rules, is that if a specific or general case doesn't trigger for a given data set, it is easy to simply write the rule that covers that particular problem, thus boosting the percentages. Having the system flag above 90% of the incorrect sentences with improbable rules is not an especially noteworthy or difficult task.

The random comparison algorithm was trained on approximately half of the translated Proust corpus from the Gutenberg Project, which totals approximately 100,000 words. For comparison, the system was trained for grammar rules containing two, three, four, five, and six POS tags (rules of size 2-6). Then the system was run on two grammatically correct paragraphs and a short message obtained from `http://faculty.washington.edu/sandeep/check/demofile.doc` that goes through Microsoft Word 2002 without causing any error messages. While the random system does trigger on both documents, it triggers at a significantly higher rate for the grammatically incorrect document, although it still misses a lot of rules.

Figure 2 shows a comparison of the number of rules triggered for a grammatically correct document versus a grammatically incorrect document. The results for Figure 2 are encouraging, as they suggest that the algorithm, despite not being perfect, has actually done something. For this graph, all of the duplicate rule triggers have been removed. At all levels, except for six where both are zero, the grammatically poor document has more triggers

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

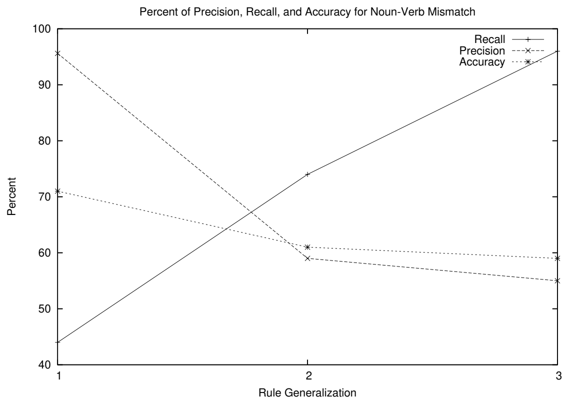

Figure 1: This graph shows precision and recall that each generalization of rule produced. On the x-axis, specific rules are 1, general rules are 2, and improbable rules are 3.

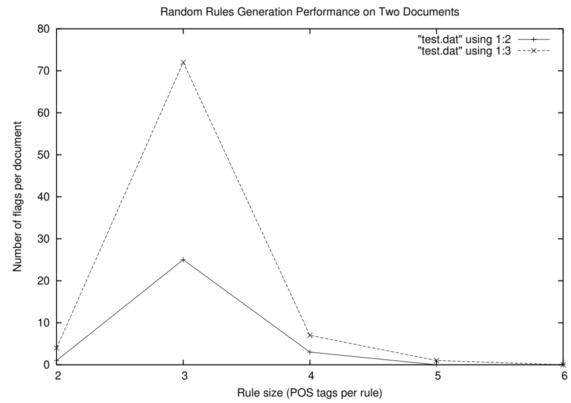

Figure 2: Random rules performance on two documents of 291 words. The upper line is the number of rule triggers for an intentionally incorrect message. The lower line is the number of rule triggers for two normal paragraphs of correct text.

than the grammatically sound document. So, while this method of random generation may not isolate rules, it may be a fairly decent measurement of the overall correctness of the grammar in a given document.

6 Conclusions

Grammar checking is not a simple problem. The Granska system works for two specific grammatical errors in Swedish, detecting them rapidly and accurately, and the two systems that were referenced earlier each had various problems that made them somewhat suboptimal. By writing out rules by hand, this system achieves results that are directly proportional to the number and accuracy of the rules that are written for a given problem. Some problems require more rules than others, but in order to hit every possible grammatical error this way, it is necessary to construct an unrealistic number of rules. A simple problem like noun-verb disagreement took this system 35 rules: 20 specific, 10 general, and 5 improbable. Describing something more complicated such as passive voice, or something more nebulous such as run-on sentences would require many more rules. On top of that, it is nearly impossible to tell if all of the rules of a given problem have been defined. On the other hand, the second random comparison method is dependent on many factors, such as the quality and size of the corpus that it's training on.

In conclusion, while it is possible to use a straight rules based system to create a grammar checker, it requires a large number of resources to create all of the rules necessary to properly define grammar problems. Using a random method obscures the rule creation process, but hopefully generates a rules set that has some bearing on what is grammatically incorrect. This system's random algorithm has a tendency to generate terminal cases that don't help define a grammar. By modifying the algorithm to include some stochastic processes, it may be possible to make the algorithm substantially better.

7 Future Work

The random comparison algorithm in this paper is fairly simple and could definitely use some adjustment. Currently, all rules that do not trigger on the training corpus are used while the rest are culled. This is a completely arbitrary decision, and it may be more effective to use a threshold greater than zero. Also, training from the Proust corpus was perhaps not the most efficient way to check for grammatically correct English. Generating rules from the Brown corpus and then testing them could generate very different results.

Finally, the main issue with a rules-based system is a lack of good ways to test it, short of having people purposefully write grammatically incorrect sen-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College

tences and manually test them. This is both tedious and of limited use. There are a relatively few corpora that are intentionally incorrect and although the knowledge that the grammar checker won't misfire is useful, manual construction of rules can only be viable if there is a good body of data to test them on. Therefore, a good extension to this project would be to attempt to generate corpora that include grammatically incorrect sentences along with correct ones for the system to train on.

References

E. Bender, D. Flickinger, S. Oepen, A. Walsh, and T. Baldwin. 2004. Arboretum: Using a precision grammar for grammar checking in CALL. In *Proceedings of the InSTIL/ICAL Symposium: NLP and Speech Technologies in Advance Language Learning Systems*.

E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. 1993. Equations for part-of-speech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789.

R. Domeij, O. Knutsson, J. Carlberger, and V. Kann. 1999. Granska – an efficient hybrid system for Swedish grammar checking. In *Nordic Conference of Computational Linguistics*, pages 49–56.

S. Sofkova Hashemi, R. Cooper, and R. Andersson. 2003. Positive grammar checking: A finite state approach. In *CICLing-2003: Conference on Intelligent Text Processing and Computational Linguistics*.

K. Toutanova and C. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*.

K. Toutanova, D. Klein, C. Manning, and Y. Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252–259.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 14–19
Computer Science Department, Swarthmore College