International Scholarly Research Network
ISRN Artificial Intelligence
Volume 2012, Article ID 918362, 6 pages
doi:10.5402/2012/918362

*Research Article*

# A Smart Proofreader for All Natural Languages: Achieving Semantic Understanding by Majority Vote

**Kai A. Olsen**[1, 2]

[1] *Department of Economy, Informatics and Social Science, Molde University College, P.O. Box 2110, 6402 Molde, Norway*
[2] *Department of Informatics, University of Bergen, 5020 Bergen, Norway*

Correspondence should be addressed to Kai A. Olsen, kai.olsen@himolde.no

Received 27 October 2011; Accepted 17 November 2011

Academic Editor: C. Gentile



The language tools offered in common word processors use dictionaries and simple grammatical rules. They cannot detect errors such as a wrong preposition, interchanged words, or typos that result in a dictionary word. However, by comparing the user's text to a large repository, it is possible to detect many of these errors and also to suggest alternatives. By looking at full sentences, it is often possible to get the correct context. This is important in detecting errors and in order to offer valuable suggestions. These ideas have been implemented in a prototype system. We present examples in English and Norwegian, but the method, that of following a "majority vote," can be applied to any written language.

## 1. Introduction

A semantic understanding of natural language is not easy to achieve. It is painfully clear that both dictionaries and grammar descriptions are far from being complete. New words emerge on a daily basis, and old words are changing their meaning. Even works of famous authors have sentence structures that violate grammatical rules. Thus, natural language computer systems based on dictionaries and grammar structures can only take us part of the way. Many have argued that one needs to be a human being in order to understand natural language, that is, to grasp all the underlying context information that is so important in getting the right interpretation. "See you at lunch" may be clear to a coworker, while the computer will ask: when, where, what, and why?

Still we have language tools that aid us in proofreading. A simple spelling checker that looks up every word in a dictionary finds many typos and misspellings. A grammar checker, even with current limitations, is an aid when we write in a foreign language. For example, the English grammar checker in Microsoft Word may find some of my is/are and has/have errors, but not all. It detects "Apples and oranges *is* used as dessert" but not "Apples and oranges, the latter imported from Spain, *is* used as dessert." And, of course, since these tools only work on a lexical and a simple syntactical level, they will not catch subtle syntactic errors, nor semantic errors. Thus, the sentence "They lived *at* London" stands uncorrected, just as "A forest has many *threes*" and "I have a *far*." To be able to find these errors we need smarter systems, that is, systems that can detect semantic errors [1].

To achieve this we need to give systems some of the understanding that we have as humans, which is so important when working with natural language. A simple method is to compare our text with that of others. That is, we can achieve some sense of semantics by comparing the user's sentences to those in large repositories. If the occurrence of "I live *in* London" is more common than "I live *at* London," then the system could suggest a change.

This idea of following the majority is, of course, not new. On a way to a soccer match in London I managed to get on the right train. From then on it was not necessary to ask for directions, I just followed the crowd. We also use this sort of "cumulative knowledge" when we follow trails, where the aggregate of many feet is manifested as a dent in the vegetation. In all these cases there may be individuals that leave the trail, do not follow the crowd to the stadium, or write differently, but these will not make a statistically significant impression [2].

That is, while it is difficult to use the wisdom of individuals, we may find an aggregate wisdom of all. This is what Brian Christian calls a "human paste" in his book on the

Turing test [3]. This "paste" offers surprisingly good results in many situations. Bots, robot programs that act as if they were humans, use this technique to give the answer to a question as a human would have done. A common method for mimicking humans is to store previously encountered answers, all the input the bot has received from users, in a large database. By using statistics they can often give (i.e., "choose") surprisingly good answers. By maintaining the conversation with a user, the bot will get new feedback that will improve its repository of answers.

Natural language translation started out by implementing dictionaries and grammar, but Google astonished the competition when they presented a cut and paste solution. Google uses large repositories of text that are available in many languages. Good sources are the UN and EU, where humans translate the same source document into multiple languages. In short, what Google has to do is to compare the user's text to these documents and then find the appropriate translation.

While these shortcuts to intelligent natural language processing can offer good results in many situations we need to be aware of the pitfalls. The cut and paste solution, or choosing an answer from statistics, does not really catch the semantics. Thus, bots work on a sentence-by-sentence basis, but do not "understand" the progress of the conversation. Brian Christian offers an example:

> User: Do you have a boyfriend?
>
> Bot: Not at the moment, but I hope to find one soon.
>
> User: I would like to be your boyfriend.
>
> Bot: That would be difficult, since I'm happily married.

Similarly, Google translation, as any other automatic natural language system, is far from guaranteeing correct results.

However, using majority vote for proofreading should be easier since the user has provided at least a first version of the sentence structure. While proofreading has many implications and can include everything from correcting a few commas and spelling mistakes to include advice and corrections on subject knowledge and organization [4], we will here concentrate on spelling, grammar, and the correct use of words, that is, enhancing the functionality of current spelling and grammar checkers. Such an application will be especially advantageous for users that write in a foreign language.

We shall present a prototype system that applies this technique. It works on all languages but is presented here with repositories in English and Norwegian.

## 2. Proofreading Using Text Repositories

In a 2004 paper [5] we described how Google could be used as an oracle for getting feedback on correct spelling and grammar, also on a semantic level. The idea was that instead of finding another person that could help, we would ask Google, that is, ask millions for advice. For example, we may wonder if it is "in," "on," or "at" "the west coast." By offering

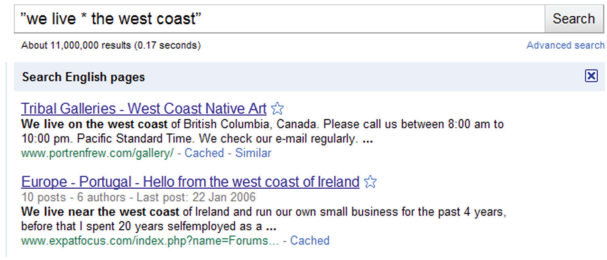


Figure 1: Using a wild card (*) in Google.

the alternatives to Google, using exclamation marks to get a frequency count for the complete phrase we will find

(i) "we live at the west coast"—2 occurrences.

(ii) "we live on the west coast"—4,240,000 occurrences.

(iii) "we live in the west coast"—7,680,000 occurrences.

In this case we can conclude that both "in" and "on" can be used here but may end up with "in" since this seems to be the most common preposition in this context.

If the alternatives are unknown, a wild card may be used in the query. An example is offered in Figure 1. The possible alternatives can then be extracted from the search engine result pages, and the method described above applied to get the number of occurrences with each alternative word in the sentence. This is a tedious process to perform manually. In addition, we have to find the phrases where we are uncertain. Many write "we had ice cream for desert" and are quite happy with their spelling.

In this paper we shall describe a prototype system that automates this process. That is, we will present a system that will find and offer suggestions for corrections for many types of errors, from spelling mistakes to semantic errors. Instead of using Google or another search engine, we will build our own text repository. This has the advantage that we have full control over the underlying data, for example, to correct punctuation and numerical errors. Also, Google and other search engines often block programmable access.

*2.1. Statistical Methods.* The method that we describe here has much in common with statistical models based on the probability that one word follows another or that a word precedes another. These probabilities may help to detect if a word is used in a strange context and to suggest more common alternatives.

Statistical models work on sequences of two words (bigrams), three words (trigrams), or N words (N-grams), for example, see [6] or [7]. The basic probabilities are computed from a text corpus. Mudge [8] uses this method to offer proofreading as a software service using bigrams and, to some extent, trigrams; Asonov [9] uses bigrams to detect typographical errors while Guo et al. apply N-grams to sentence realization [10]. The statistical method has the advantage that some of the processing, that is, finding the probability values, can be performed up front. However, most statistical methods work on sequences of only two or three

words while the method that we propose here works on full sentences. However, note that the two methods will be similar if all words, all punctuation, and complete sentences from the sources are stored in the repository.

*2.2. Working Directly on the Text Repository.* The advantage of working directly on the text repository, instead of going via statistical methods with a limited number of word combinations, is that the more of the context that is considered, the better suggestions for improvement that can be made. While most statistical methods work on two- or three-word combinations, we can now consider the complete sentence given by the user. Another advantage of working directly on the repository is that, in principle, new text can be added consecutively, for example by a spider that traverses the Web or bibliographical databases.

The disadvantages of this approach are that we need large repositories in order to find data on the complete sentence, or large parts of this. Demands for processing during the analysis phase will, of course, be much higher since we do not perform any significant preprocessing. Instead of looking up words and word combinations in a database we will need to process large parts of the repository, at least large index structures. However, this can be tackled by modern computer technology as demonstrated by Google and other search engines. With multimillion clusters of computers they can handle complex searches in just a few seconds and, as we have seen, also language translation that requires traversal of large text or index repositories.

## 3. The Prototype

In order to test our ideas we have developed a simple prototype system (Figure 2). This consists of three parts:

(1) a spider that creates the text repository,

(2) a builder that creates the indexes we need for fast processing,

(3) an analyzer that analyzes the sentence offered by the user, suggesting improvements.

The spider is given a seed of several Web addresses, to the sites of universities, newspapers, government institutions, and so forth. It will then download everything from each address, retaining the text part and "clean" the text by removing formatting commands. The spider will also collect all the links that it finds on a page. These are stored for later processing so that we avoid making too high demands on one site at a time. The final product is a collection of sentences that are stored in sequential numbered files in the repository, each of one megabyte. The one megabyte size was determined by an estimate based on the time to parse the file and the size of the index structures. Larger files take longer times to parse, but will also limit the size of the indexes.

Each link (URL) is represented by a hash value. These are stored in a table of visited sites. If the same URL comes up in another setting, we can avoid a second parsing of the site. The spider worked for several weeks to collect a 2.5 gigabyte repository in English (.com-, .uk-, .org-, .edu-sites) and a 1 gigabyte repository in Norwegian (.no) sites. A better alternative than to use extensions would have been to detect language automatically. That is, we may find text in other languages than English on .com sites and other languages than Norwegian on .no sites. However, the occurrence of this "noise" is seldom statistically significant.

Table 1: Index structure (example).

(a)

| Word | Occurrence in file |
|---|---|
| Ice | 4, 17, 54, ... |
| Cream | 17, 98, 109, ... |
| Dessert | 2, 17, 234, ... |
| ... | |

(b)

| File no. | Word | Occurrence in lines |
|---|---|---|
| 4 | Ice | 967, 1201 |
| ... | ... | |
| 17 | Desert | 32, 2378 |
| 17 | Cream | 567, 1201, 1456 |
| 17 | Dessert | 1201 |

Since we only had one PC available for this research project it became necessary to create an index structure for fast access to the repository. As seen from Table 1 we maintain a table with all words found in the repository, giving a list of files where each word is found. Each line in the table, that is, word and file references, is stored in a simple .txt file, using the word as the file name. Then, for each file and word, a new table offers a list of the sentences within the file where the word appears. These references are also stored as simple files.

For the Norwegian version all words were stored in the index, giving a number of approximately 600.000 words. Note that this includes typos, names, and word combinations (in Norwegian, words are often tied together, such as in "skistaver"-ski poles). In an effort to avoid spelling mistakes in the English version, we only stored words that had an occurrence frequency greater than ten. This gave us approximately 226.000 words in English.

The idea is that when the user offers a sentence with N words the analyzer can limit parsing to the files and sentences where all these words occur. However, since we also try to find alternatives, we actually have to parse all sentences where N minus one of the words occur. We implement this by replacing the first word in the sentence by a wild card, collect all alternatives that may be used for this word, do the same with the second word, and so forth.

An example of this analyzing process is presented in Figure 3. The user's sentence is "I live at London." From the intermediate part we see that the analyzer has found zero occurrences of this sentence but has found four where "in" replaces "at." Note that only words that are similar to what offered by the user are considered. The idea is that users often exchange words with a similar spelling. We use Hirschberg's

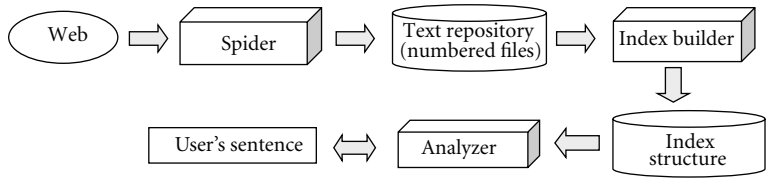


Figure 2: Spider, index builder, and the analyzer.

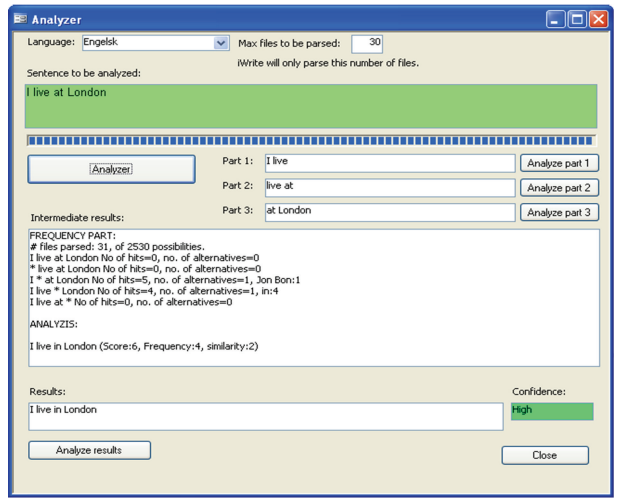


Figure 3: Screen shot of the analyzer (example 1).

algorithm to compute this score value [11]. In addition, since many users interchange prepositions, these are also considered "similar."

The difference of occurrence of the "at" and "in" alternative is great enough (0 to 4) for the analyzer to propose the results "I live in London" with a high confidence.

In another example, Figure 4, the analyzer finds the correct spelling, changing "desert" to "dessert," but since the dessert variant only occurs twice in the repository, the confidence is marked as low.

The prototype handles many different types of errors. It corrects prepositions, typos, interchanged words, and facts. Still, it has several drawbacks. Computing time for the analysis process, even with the elaborate index structure, may be everything from a few seconds to more than a minute.

It also becomes clear that the repositories are too small. This is especially the case for the one gigabyte Norwegian database. For example, it corrects "Jeg bor på Oslo" (I live at Oslo), but not "Vi bor på Oslo" (We live at Oslo), since it cannot find sufficient data on this variant. But even the 2.5 gigabyte English database is way too small. There are constructs that are not found at all and others where the frequencies are too low to give significant results. We also find that the Web is not a good source for finding quality text. Even when we eliminate all words that have a frequency of occurrence less than ten, we end up with a lot of spelling mistakes and typos.

A product version of the proofreader would have to use a large cluster of machines, where each single computer could parse only a fraction of the database, perhaps only one file [12–14]. By working directly on the repository, indexes can be avoided. To improve the quality of the repositories, a better alternative is to start with quality documents, for example, the UN and EU sources that Google use. In addition, bibliographical databases with books, reports, scientific papers, and so forth could be useful. However, it is a problem that we need really huge repositories.

We can envision such a tool as a site where we can upload documents for proofreading. Better, the tool should be integrated with the word processor and offer concurrent advice, just as contemporary spelling and grammar checkers.

## 4. Semantics by Following the Majority

Statistically we make a "paste" of all the examples in the repository, following the majority vote. Is this what we want? The answer will be dependent of the user's aims and knowledge. If we are proficient writers, a system such as this may help us to find some typos, but apart from that it could be annoying-underlining sentences that it found suspect, indicating errors that may not be there. However, when we write in a foreign language the idea of following the majority will be great for most of us, especially if this can aid in avoiding blunders. From the simple tests we have performed on the prototype it is clearly of help to nonnative writers, finding errors similar to the examples that have been presented here, that is, if one has patience with the processing and repository limitations of the prototype.

What all current proofreading systems, including the system we have presented here, really do is to move experience

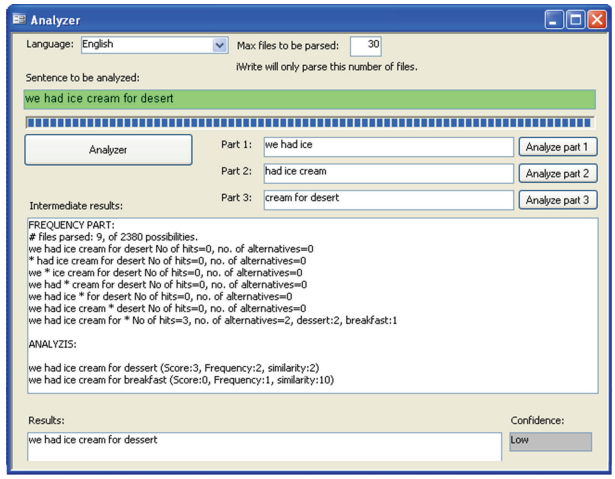


Figure 4: Screen shot of the analyzer (example 2).

of writing to that of reading, that is, moving from recall to recognition [15]. Since most of us are better readers than writers, we may be able to determine which suggestions to follow. If the sentence provided by the user has a zero or low frequency in the repository, the system may come up with another choice for alternatives that occur more often, especially if the words have a similar spelling. For example, "I visited New Fork" (a ghost town in Wyoming, USA) may be suggested to change to "I visited New York." Thus we must rely on the user to ignore these false indications, comparable to being on the train that takes supporters to a soccer game, but where we have a different destination. This is, of course, crucial to all language aids. We depend on the user to make the right choice.

To get the correct meaning of the user's sentence we should use as much of the sentence as possible. If we study only one word at a time, there may be many different interpretations of each word. With more words, especially full sentences, the context becomes well defined and it becomes easier both to find errors and to offer valuable suggestions. Since nobody (in the repository) "has a red far," we may suggest "car." But "we live at the West Coast" may look like an error, until we unveil the next word in the user's sentence that may be "hotels," that is, "we live at the West Coast hotels" is correct. The problem is, of course, that if we analyze long sentences there may not be any occurrences of the complete sentence in the repository. We then have to look at parts, but with the danger of loosing context.

The prototype analyzes a sentence at a time. It may look into parts if the repository cannot provide data on the complete sentence. However, a product version may also need to analyze more than one sentence at a time. For example, this will be necessary to catch the reference error here: "The house was sold. He went for a million." There may be types of errors that require the proofreading system to look into whole paragraphs, perhaps the complete text to make valuable suggestions. For example, as we have seen a nonnative English speaker may easily exchange "desert" or "dessert." While the sentence in question may be "The desert was impressive," we may need to analyze more of the text to see if we are in a "desert" or "dessert" context. These cases can also be solved statistically, for example, by studying how other words in the text go together with each of these alternatives. Hopefully, the user writes about deserts *or* desserts, not both at the same time.

## 5. Future Work

A limitation of the current version of the prototype is that it tries to replace one word with others. However, often the problem may be that there are unnecessary words in a sentence offered by the user. As an example, consider "All the information is sent to PayPal and they are who manage the collection." An abbreviated version is better: "The information is sent to PayPal, which manage the collection." This functionality, that of removing words or replacing a set of words with one word, would be cumbersome for the prototype due to a limited repository but should be quite easy to implement in a product version.

It is expected that the user provides a sentence that has a reasonable well-formed structure. The system may suggest alternative words and remove words, with the functionality discussed above, but cannot change the structure using the simple comparison method that we have suggested here. This may limit the value of the system for nonnative writers that often mess up the correct order of words. As a consequence of this we may also find that there may be no data in the repository for the incorrectly structured sentence, thus even limiting the systems capabilities of correcting single words.

As an example, consider "We can see in the figure the class diagram for the website," written by a student using English as a second language. Clearly, a better alternative would be "The figure shows the class diagram for the website" or "The class diagram for the website is shown in the figure." However, in both of these improved sentences the superfluous word "we" have been removed, in addition to a new sequencing of words. It is outside the scope of this paper to find solutions to this problem and is therefore offered as an interesting topic for future research. An option could be to provide alternative sentences from the repository that

had a large fraction of the words from the initial sentence. However, superfluous words, such as the "we" above, may be an obstruction.

## 6. Conclusion

We have presented an idea and a prototype implementation, of a proofreading system that can find and suggest corrections for many types of errors, also semantic errors. The idea is to compare the user's text to a large repository, ideally trying to find better variants for each word where the original sentence is not found in the repository. Thus, it is possible to catch the right context.

While the prototype works on limited repositories and is too slow with its one computer installation, a production system needs large, high-quality repositories and a multicomputer setup. In order to be able to improve also the structure of the sentence provided by the user, an enhancement of the basic method is needed.

## References

[1] C. S. de Souza, *The Semiotic Engineering of Human-Computer Interaction, Part I*, MIT Press, Cambridge, Mass, USA, 2005.

[2] K. A. Olsen and A. Malizia, "Following virtual trails," *IEEE Potentials*, vol. 29, no. 1, pp. 24–28, 2010.

[3] C. Brian, *The Most Human Human: A Defence of Humanity in the Age of the Compute*, Penguin Viking, 2011.

[4] N. Harwood, L. Austin, and R. Macaulay, "Proofreading in a UK university: proofreaders' beliefs, practices, and experiences," *Journal of Second Language Writing*, vol. 18, no. 3, pp. 166–190, 2009.

[5] K. A. Olsen and J. G. Williams, "Spelling and grammar checking using the web as a text repository," *Journal of the American Society for Information Science and Technology*, vol. 55, no. 11, pp. 1020–1023, 2004.

[6] I. Langkilde and K. Knight, "The practical value of N-grams in derivation," in *Proceedings of the 9th International Workshop on Natural Language Generation*, pp. 248–255, New Brunswick, NJ, USA, 1998.

[7] H. Nakanishi, Y. Nakanishi, and J. Tsujii, "Probabilistic models for disambiguation of an HPSG-based chart generator," in *Proceedings of the 9th International Workshop on Parsing Technology*, pp. 93–102, Vancouver, British Columbia, Canada, 2005.

[8] R. Mudge, "The design of a proofreading software service," in *Proceedings of the NAACL HLT Workshop on Computational Linguistics and Writing*, pp. 24–32, 2010.

[9] D. Asonov, "Real-word typo detection," in *Natural Language Processing and Information Systems*, H. Horacek et al., Ed., vol. 5723/2010 of *Lecture Notes in Computer Science*, pp. 115–129, 2010.

[10] Y. Guo, H. Wang, and J. van Genabith, "Dependency-based n-gram models for general purpose sentence realisation," *Natural Language Engineering*, vol. 17, no. 7, pp. 455–483, 2011.

[11] S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[12] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating System Design and Implementation*, San Francisco, Calif, USA, December 2004.

[13] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[14] J. Dean and S. Ghemawat, "Map Reduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[15] J. Nielsen and R. Molich, "Heuristic evaluation of user interfaces," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '90)*, pp. 249–256, Seattle, Wash, USA, April 1990.