N-gram Based Grammar Correction

1 Problem Statement

Spelling and grammar errors infest the lives of everyone from native to non-native speakers of a language. We will consider the problem of spelling and grammar correction. Traditionally, spellchecking utilizes a dictionary, but this creates the problems of out-of-vocabulary and data sparseness. To overcome this many have used a corpus, a database of existing texts, usually hundreds of books and newspapers. These corpora can be tagged or untagged. A tagged corpus contains, for each word, a tag describing which part of speech the word represents. The corpus is usually preprocessed into counts of n-grams. An n-gram is a sequence of tokens, usually words or characters of length n. Different approaches to solving the problem include using tagged or untagged corpora and using n-grams with different n in different parts of the error correction process. In a related field, machine translation does not traditionally utilize much language data (except for the translation model), which [4] suggested could improve translation. This suggests a naive improvement for machine translation: run a spell checker on poorly translated text. We have considered using an untagged corpus and counts of combinations of different sized word punctuation n-grams to check for spelling and grammar mistakes in texts and translations, and offer corrections based on these counts.

2 Related Work

Traditional techniques include using a language model including a grammar model of the language. Relatively recently ([3] published one of the first datasets in 2006) other techniques using probabilistic models of language with much bigger datasets have started taking over and offered much better accuracy.

The technique described by [2] uses 3-grams consisting of words and parts-of-speech and checks for errors based on their probability of occurrence in a tagged corpus. This method is insufficient because it doesnt vary the size of the n-grams. It also incorporates part-of-speech, which increases permutations and complexity.

The algorithm by [1] is comprised of a spelling error checker using unigrams information from Yahoo! N-Grams Dataset; a candidates generator based on a letter-based 2-gram model; and a context-sensitive error corrector that selects the best spelling candidate for correction using 5-grams information from Yahoo! N-Grams Dataset.

3 Implementation Details

Our implementation is based on known approaches utilizing three phases:

We implemented a spelling and grammar checker using an n-gram database. We created the database by extracting n-grams with n equal to 1 through 5 and scored them based on frequency in the corpus using the 1-million word Brown corpus from the Natural Language Toolkit. First, trigrams were used for correction. For this, we extracted all trigrams in each sentence and found a confusion set of words that each middle word in the trigram could be confused with. To do this we searched the database of scored trigrams using a merge algorithm to find all trigrams that were identical to those of the trigram in the sentence being corrected except for the middle word. Here is an example of confusion set search for the trigram

He went home

Merged trigrams in the database would be of the form:

He ran home
He was home
He walked home

So the confusion set would be: {*ran, was, walked*}. Candidate sentences were then generated using the confusion set for each word in a sentence. Next, the Damerau Levenshtein distance algorithm was used to determine how similar each word in each candidate sentence was to the corresponding word in the original text. The three candidate sentences with the least distance and highest score are returned. These candidate words are then combined into candidate sentences.

The second phase is to score each candidate sentence by creating uni-, bi-, tri-, 4- and 5-grams from the candidate sentences and then score those scored trigrams based on the n-grams from the corpus.

The final phase is to put these scores together and select the best candidate. For each sentence we summed each set of n-grams to generate a total score for each type of n-gram. This was done for each original sentence and its candidate sentences. We used the same technique to calculate the total edit distance to the original sentence and used that as a score too. A linear interpolation of these scores using manually adjusted weighting produced final evaluation values used to select the correct sentence. After performing this evaluation and selection for all the sentences in the documents, The corrected document was compared to the original document using word error rate (WER).

4 Evaluation

For evaluation, we used two methods. The first method evaluated our improvement of a poor translation. We constructed the poor translation by taking a document manually translated to traditional Chinese and using Google translate, translated this document to English. We then compared this text using WER (Word error rate) to the original English version of the document to get a base score. To evaluate our algorithm we ran this poorly translated text through our algorithm and compared the results with the original document using WER, which is the percent difference between the documents. We did this using two different methods of finding errors, using just trigrams, and using n-grams with n equal to 1 through 5. We put this translated document into Microsoft Word, corrected errors if suggestions were given and compared this to the original document using WER. The results are shown in Table 1.

Documents Being Compared	Percent Different	
	Using 3-grams	Using 1-5-grams
Translated GPL vs. GPL	61%	61%
Corrected GPL vs. GPL	68%	61%
M. Word GPL vs. GPL	66%	

Table 1: Original text vs. corrected text comparisons

The second method we used to evaluate our approach was trying to correct errors introduced into a text. We did this by choosing *n* sentences randomly in a test text and then introducing one error per sentence. We then ran those sentences through our algorithm and counted the number of correctly corrected sentences and uncorrected or falsely corrected sentences. Our average percentage of correct sentences was 72%, with 28% divided between errors not corrected and falsely-corrected words. [1] used this same approach to test their algorithm and came up with impressive results due to their use of a 3.4-billion word corpus tagged with parts-of-speech, whereas [2] used a corpus of 1-million words. The results of [1] and [2] are shown in table 2

Types of Corrections	Successfully Corrected	Not Corrected	Falsely Corrected
Ours	72%	16%	16%
Bassil[1]	94%	2%	4%
Fossati[2]	50%	32%	14%

Table 2: Results of correcting errors introduced in a text

The results above make it evident that a very large corpus increase precision drastically. It should be noted though that the evaluation method for our algorithm as well as that of [1] used a document with simple errors. Further testing could lower all the percentages in table reftext.

5 Discussion

A larger corpus would definitely have helped, given a powerful enough computer to handle it. The use of a small corpus tends to produce worse results because of data sparseness, which gives rise to situations where the correction does not exist in the corpus, making such corrections impossible without a grammar model. From our first experiment, we learned that using a linear combination of the scores from n-grams where n is equal to 1 through 5 our precision increased. This is due to the fact that larger n-grams can include two or more words that are grammatically dependent, which only using 3-grams cannot due. The results in table 2 show that software such as Microsoft Word and almost all other do a very poor job at performing grammar correction. Something to conclude from our testing is that software such as Google Translate and Bing Translate probably already utilizes the same information found in a corpus to do this sort of check during their translations, and therefore our effort becomes redundant and ineffective. By training the weighting of the scores for each type of n-gram our algorithm could have produced improved results. [1] generated a confusion set by using bigrams of characters and evaluating those. We used a different approach, that while being more complete, was also slower. The slowest part of our algorithm was actually choosing the confusion set for each character by measuring the edit-distance between the original word and the candidate. Using the faster Levenshtein distance instead of the DamerauLevenshtein distance we achieved a considerable speedup. Overall we learned that using solely n-grams to correct grammar is insufficient, and the use of some property of grammar is necessary, whether it be a part-of-speech tagging or some other method. If we were to change something it would be to use a grammar model. The effect of using an extremely large database could be investigated as well, but a grammar model would most likely be more accurate. Also, especially in the area of foreign language translation, semantic modeling would be interesting to investigate.

References

[1] Bassil, Y. Parallel spell-checking algorithm based on yahoo! n-grams dataset. *Science 3*, 1 (2012).

[2] Fossati, D., and Eugenio, B. D. A mixed trigrams approach for context sensitive spell checking. In *Computational Linguistics and Intelligent Text Processing* (2007), vol. 4394, pp. 623–633.

[3] Michel, J., Shen, Y., Aiden, A., Veres, A., Gray, M., Pickett, J., Hoiberg, D., Clancy, D., Norvig, P., Orwant, J., et al. Quantitative analysis of culture using millions of digitized books. *science 331*, 6014 (2011), 176–182.

[4] Norvig, P. Natural language corpus data. *Beautiful Data* (2009), 219–242.