Spelling Checker with a Trigram Language Model

Matt Cotter
NLP, Fall 2013

I. Problem Description

My final project was to create a spelling checker that would offer good suggestions and mark known words to be misspelled if they were used incorrectly. Typed text is everywhere in the world, and more and more textual content is being created electronically everyday. Whether the error is a typo, an incorrect knowledge of the word, or a difficult word to spell, misspellings happen quite often. Not only are spelling errors unprofessional, they also can make text impossible to understand.

Due to all of these reasons, a good spelling checker is vital. Though a spelling checker was one of the earlier homeworks, that original assignment for class had several shortcomings that I have sought to improve. These improvements include weighed letter substitution cost, a trigram language model, and a dictionary trained from a corpus.

II. Corpora

For my training my trigram language model and dictionary, I used several NLTK corpora. The first was the Gutenberg corpus. This consists of text files for eighteen books and has over two million words. The second corpus used was the Reuters corpus, consisting of ten thousand news documents with around 1.3 million words. The final corpus used in training my model was the Brown corpus. This has excerpts from fifteen different genres and around 1.15 million words.[1]

For testing my spelling checker, I used the Holbrook corpus. This corpus consists of excerpts from the book *English for the Rejected*.[2] The text was written by "secondary-school children, in their next-to-last year of schooling".[3] This is a corpus of text that has some errors tagged. These included errors in capitalization, conjugation, splitting up or combining words, and errors in spelling. No distinction was made between error type.

III. Methods

I started by training the counts for a trigram language on the training corpus. Additionally, I trained a dictionary of encountered words, and a hash map from single letters to a list of all words that

1 http://nltk.org/book/ch02.html#tab-corpora
2 http://education.cambridge.org/us/subject/english/english-for-the-rejected
3 http://www.dcs.bbk.ac.uk/~ROGER/corpora.html

begin with that letter.

I then manually created a symmetric mapping from all letter pairs to their corresponding substitution costs. While doing this I accounted for horizontal, vertical, and diagonal adjacencies on a qwerty keyboard in addition to letter similarity (by English pronunciation).

The next step was to implement the spelling checker. Each sentence in the input text is tokenized into disjoint strings (words, symbols, or whitespace) that can be concatenated to reform the original sentence. A second token list is then created of only the words. This list is traversed, and each word is examined. Every word is assigned a probability, which is an average of the trigram probability of the two preceding words and itself, and the preceding word, itself, and the following word. These trigram probabilities are each calculated via a weighted linear interpolation of unigram, bigram, and trigram probabilities, calculated as a quotient of counts.

Every word is also assigned a list of best matches. To get the best matches, first the set of candidates is identified. This set is every word that was seen following the last two, and every known word that starts with a letter equal, “similar”, or horizontally adjacent to the first letter of the word in question. The “score” value given to any candidate is a linear combination its string edit distance of the word in question, and the probability it would have if the replacement occurred. The list of best matches is the top 10 candidates by score.

Finally, if a word has a low probability and is not itself one of the top three best matches, or has an extremely low probability and is not itself the single top best match, it will be marked as “incorrect”. When run outside of testing mode, the user will be prompted to fix the marked word with a list of the best matches as options.

IV. Problems Encountered

A problem I encountered immediately was tokenization. In order to meaningfully be able to repair input, the token list had to be able to be used to reconstruct the initial sentence in its entirety. This included preserving whitespace and avoiding introducing whitespace between adjacent tokens (for example between a word and a comma).

Another problem encountered was the slow processing rate. As an interpreted language, python is inherently slower than something like C or even Java. However, after analyzing my code with a profiler and optimizing my process, I gained a significant speed up. There is no longer much of a noticeable lag time between word suggestions, and my entire training corpus can be analyzed in under

2 hours on my laptop. Despite these speedups, loading the language model into memory will always take a very long time (~18.00 seconds) unless it is made smaller.

One more difficult problem I encountered was scoring each candidate when finding the list of best matches. I had to first find the best way to calculate the which words are likely from the language model, then I had to figure out how to weight the probabilities against the edit distance. My calculation of the probabilities was informed by my results from the author classification assignment. The weights of the probabilities against the edit distances were determined via repeated trials and observing trends in whether or not recommendations were dominated by similar words contextually or similar words by spelling.

V. Results

After running my spelling checker on my training corpus, I divided the identifications into six categories: 1) correctly identified as not misspelled, 2) incorrectly identified as misspelled, 3) identified as correct, but the word was misspelled, 4) correctly identified as misspelled. Category 2 was split into 2a) words that appeared in their own list of best suggestions, and 2b) words that did not. Category 4 was likewise split into two categories 4a) the correct spelling of the word appeared as a suggestion, 4b) the correct spelling did not appear as a suggestion. The counts are as follows:[4]

Category	Counts
1	20744
2	2301
(2a	1301)
(2b	1000)
3	645
4	1586
(4a	914)
(4b	672)

From this table, you can see that out of these correctly spelled words, a little more than 90% were marked as correct. Of the incorrect words, slightly over 70% were identified as misspelled. However, many of the incorrect words were not misspelled, they were simply used in the wrong context. Many of the words the checker could not fix were proper nouns, as were many of the words spelled correctly that the spell checker identified as misspelled and couldn't offer a good suggestion for.

4 See “output.csv” for full results

VI. Future Improvements

There are several ways that this project could be improved with future work. The most important improvement would be a better test corpus, especially one more similar to the training corpus. Another improvement would be somehow trimming down on the memory size of the language model. A more functional improvement that would help a lot would be the ability to correct a single word to multiple or multiple words to fewer. A final improvement could be training the substitution costs rather than assigning them.