Billion Word Imputation – Project Proposal

Abstract

In this proposal, we will detail our group's plan to solve the billion word imputation problem. Essentially, given a large training set of sample sentences, we wish to create an algorithm that can predict the missing word in a sentence with one word removed.

1. Size of Team

4 Members

2. Motivation

There are multiple strong motivations for attempting to solve this task centered around adding a mechanism to accurately recover or replace otherwise undeterminable data. In fields related to auto-correction, progress on this task is useful for correcting writing mistakes on both a word and sentence level. In fields related to auto-suggestion, progress on this task would affect scanning-to-text implementations, damaged textual data set recovery, speech recognition, assisting idiom or phrasing completion, and many other similar issues.

More generally, progress on this task may give insights to language structure by discovering patterns for when phrases or sets of words are strongly associable, and where one has to draw the line between using language knowledge to fill in the blank as opposed to contextual knowledge. Or it's possible that a machine learning based approach has the ability to surpass humans in word imputation, since word imputation in the above contexts can be a difficult enough problem already for humans ourselves.

3. Statement of Task

Given a sentence with one word missing, we want to be able to guess what word was removed, and where in the sentence it lies. There are many ways for us to evaluate how good the imputation really is. How close is the location of the imputed word to the location of the missing word? Did we get the correct type of word (noun, adjective, adverb, etc.)? Does our imputed word follow the rules of basic English grammar? When we propose an imputation, how confident are we that the imputation is correct?

4. General Approach

Our first avenue of approach will be to build a Markov Chain model of the training sentences that we would then use to experimentally insert words into the sentence. For every possible location, we will try inserting the words that the Markov Chain model gives a high probability for occurring in that location. If inserting a word in a particular position results in a new sentence that is significantly more likely to be created by the representative Markov Chain than the original sentence was, we may have found the correct insertion. The interesting problems with this approach are to determine the best states to use to build the Markov Chain model. For example, the state of a Markov Chain at a given position can be determined by any number of the preceding words - different length n-grams (n-word phrases) can be used as the states of the Markov Chain, and indeed 2-gram or 3-gram states would likely represent the data much better. Combinations of different values of n can be explored to - is it beneficial to sometimes use 2-grams to determine the probability of the next word, and sometimes use 3-grams?

The main problem of the Markov Chain modeling approach is that language has long-distance dependencies, and the obvious Markov Chain constructions (where states are the previous words) do not. A possible variation on this approach could be keeping track of different words for different lengths of time - when we build the Markov Chain model, the state would consist of whether a word has occurred recently, with 'recently' being defined differently for different words. This idea has aspects of a positional language model, which specifies the probability that two words occur a certain distance from each other in a body of text. Exploration could be done into whether remembering certain types of words (for example, nouns vs. verbs, or subject vs object of the sentence) for longer than other types of words improves the model. However, this introduces further problems. Part-of-speech identification can also be done by Markov Chains[1], but grammatical relations likely require natural language processing techniques.

[1] http://en.wikipedia.org/wiki/Part-of-speech_tagging

5. Updated Approach

We've since brainstormed and selected a number of specific approaches to try to implement and compare. There are a number of evaluation criteria we'll use to compare the approaches. The primary criteria is accuracy; how well do the different approaches do under Kaggle's evaluation system? We'll also be considering time and cost. Specifically we'll consider both the compute cycles of each approach as well as their Big-O runtime, and whether there's resources specific approaches need (space, external data, etc.).

5.1 N-Grams

This is the approach we originally detailed when creating the project proposal. Under this approach, we'll go through the training data and count n-grams with n = 3 or possibly higher depending on how efficient we can make our implementation. For each word W in the test data, we'll look up associated n-gram models and raise flags that there's likely a missing word before W if the word W is judged unlikely to occur based on the previous words. This can also be done in reverse, testing how likely W is to precede the following words. Word placement will be guessed based on words the n-gram model judges likely occur at these spots, and to select which word placement we would use, or if we place any at all (the original sentence might already be complete), we would calculate the total probability of each possible sentence (with possible word placements, or without any) and select the most probable. The main concern with this approach is efficiency and we're still ironing out the fine details on how to make both training and testing more efficient.

5.2 Averaged Perceptron Location Identifier

The idea here is that there may be some features we can train on that, given an empty "space" and words surrounding it, determine if it's probable that a missing word belongs in the location. Likely key features include the parts of speech of the words surrounding the "space", which we would use an external POS tagger to identify, or the brown clusters (or other clustering techniques) of the words surrounding the "space". Because the training data is of complete sentences, each piece of training data would be modified to either be complete, or have a word randomly removed. Averaged perceptron has shown to do well on other NLP problems, so we believe that it could assist in producing good results here. This technique is specific to identifying the location of the missing word, which is an interesting enough problem on its own to deserve special consideration, and if this approach produces good results it can be combined with other approaches that solve the second subproblem (identifying the missing word given a specified location).

5.3 Long-Distance Bigrams

This technique is mainly aimed at determining the location of the removed word, although it can be used for imputation as well. In this approach, we keep track of the frequency of all long-distance bigrams up to a to-be-determined distance d, as described in this article[2]. A long-distance bigram of distance d is an ordered tuple of two words, which contain (d-1) words between them. After counting the frequency of all long-distance bigrams in the training data, we can estimate the probability of one word following a sequence of other words as Pr(Wi | Wi-d, … , Wi-1) = sum from k=d to k=1 of lambda * Pr(Wi | Wi-k), where lambda is another constant to be optimized. [also cite the same article]. This particular method of building a probabilistic language model makes it very simple to guess where a word has been removed: simply insert 'wildcard' words into all possible locations in the test sentence (one at a time), and calculate the probability such a sentence occurs normally, using Pr(Wi = wildcard | Wi-k) = 1 and Pr(Wi | Wi-k = wildcard) = Pr(Wi). To use this method for imputation, we have to perform 'guess' insertions at each location based on the calculated probabilities of words occurring there due to different bigrams they would be in, and then calculate the probability of the resulting sentence based on the above model. We have long-distance bigram counting implemented for this approach, but have found it too slow to train on the entire dataset at once. We plan to modify the solution to train on one tenth of the dataset at a time, which we have found to be a manageable amount.

6. Resources

We will be using the test and training data from the Kaggle competition page[3].

We will also be using parsing libraries for Python or a similar language to parse sentences and grammar. Depending on the computational needs of the algorithm, we may also need to utilize AWS server time to complete the learning in a reasonable amount of time.

7. Schedule

Oct 21– Oct 28: Research background information and methodology for completing the task (Markov chains, n-grams, NLP, etc.)

Oct 29 – Nov 8: Create an initial imputation and test it out

Nov 9 – Nov 11: Progress report, continue working on imputation

Nov 15: Finish some imputation algorithm by this point

---

[2] http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=9F1EA89E65C54F2F27D51B46C9F774B0?doi=10.1.1.45.1629&rep=rep1&type=pdf

[3] https://www.kaggle.com/c/billion-word-imputation/data

Nov 16 – Nov 26: Refine existing algorithm to lower test error

Nov 27 – Dec 4: Analyze results and complete final poster