Document Reconstruction from a Bag of Tokens using Trigrams and 5-grams from a Corpus of Related Documents

Brad Burkman
University of Louisiana at Lafayette
Louisiana School for Math, Science, and the Arts
bburkman@lsmsa.edu

Abstract

We have a bag of tokens, perhaps from a corrupted document, and a corpus of documents known (or supposed) to be "similar" to the original document. Can we use the corpus to reconstruct a "best guess" approximation of the original document?

We propose a new method. We first use the word frequencies in the corpus to order the bag of tokens and turn it into a bag of words with a Ziphian distribution. Then we make a graph from the corpus trigrams and find a path through the graph that links tokens to build a string of words to form a candidate document.

We have scored the candidate using two metrics and plan to add a third: (a) the sum of the corpus frequencies of the trigrams in the path, (b) the sum of the frequencies of any corpus 5-grams that appear in the candidate document, and (c, not yet implemented) a measure of how closely the word distribution in the candidate document approximates the Zipfian distribution of the bag of words.

In development we used the NLTK `Inaugural` corpus, the texts of fifty-six US Presidential Inaugural Addresses (Bird et al., 2009). We plan to also test it on other corpuses.

We discuss why we believe our method, while it may be interesting, is not likely to be successful as currently envisioned.

1 Credits

This paper is a project for CSCE 561, Information Storage and Retrieval, taught by Dr. Aminul Islam at the University of Louisiana at Lafayette.

The code relies on NLTK, the Natural Language Toolkit (Bird et al., 2009), and `graph-tool` (Peixoto, 2014).

2 Introduction

The goal of Natural Language Generation (NLG) is to build sentences (which are subsequently built into conversations or documents) from some information stored in a structured form (Matulik, 2018), and it is hard. Even small improvements are important (Barros and Lloret, 2015).

The algorithms for Natural Language Generation are np-hard with $O(n!)$ because we have to consider all (or most) possible paths through the graph. If we find all walks with unlimited cycles, even in a small graph, then we have an $O(\infty)$ problem. In many other related problems we could prune branches early when we realized that the path was not going to lead to an optimial outcome, but NLG is like the game of Go, that " their outputs can be evaluated only when their process reaches the last state" (Kumagai et al., 2016).

The hypothetical application considered here is that we have a corpus of "similar" documents, and there is another similar document for which we only have a bag of tokens. Our goal is to find a "best" candidate for a reconstruction of the lost document using the corpus.

We chose to work with the NLTK Inaugural corpus, and from that generated a smaller bag of tokens from the tokens in the corpus. Following the method described below, we applied a corpus-derived Zipfian frequency distribution to turn the bag of tokens into a bag of words and worked towards generating a candidate reconstruction of the "lost" document by finding and scoring paths through the graph of trigrams.

3 Original Contributions

We believe that our scoring mechanism is novel and useful. While we have only implemented the first two stages, we believe that this set of three metrics would give a good measure of similarity

to our goal, finding the candidate that most closely represents both the bag of words and the phrasing of the corpus documents.

The first metric is the corpus frequency of the trigrams used in the candidate document. Since the candidate is constructed by linking corpus trigrams, this metric will be at least the number of trigrams in the candidate, but will favor candidates that use trigrams more common in the corpus.

The second metric is the corpus frequency of the corpus 5-grams that appear in the corpus. The algorithm constructs the candidates with no reference to the 5-grams, but since 5-grams reveal longer phrases than trigrams, this metric will give a better measure of how close the candidate is to the grammatical and oratorical patterns of the corpus.

The third metric will be the similarity between the word frequency distribution in the candidate and the bag of words.

Adding more metrics like 7-grams would push the candidates towards splicing together long phrases from individual speeches rather than taking the corpus as a whole. Also, in a corpus this small (fifty-six documents), it is unlikely that many 7-grams would have a document frequency more than one.

We believe that these three metrics together give a better measure than any of them separately. Given that we have no gold standard for evaluating the quality of the candidates, this set of metrics may be among the best options.

4 Related Work

Other work has similarly tried to build new texts word-by-word by choosing a few starting words and analyzing a corpus to see which words could follow those, building up the text iteratively. Matulik did this with a probabilistic model (Matulik, 2018). Srivastava built up a document letter by letter, based on the previous hundred letters (Srivastava, 2018).

Baptiste Fontaine, a contributor to Homebrew and other projects, has a trigrams-based random text generator on GitHub (Fontaine, 2014). It takes in some texts and returns a text, by default up to seventy words. The documentation does not indicate the method or whether there is a scoring mechanism.

5 Method

In development we used as the corpus of documents the *C-Span Inaugural Address Corpus*, fifty-six speeches available in NLTK as `Inaugural` (Bird et al., 2009).

We will illustrate the methods with results from a run on eight of the speeches. More than eight speeches made the $O(n!)$ path calculations unreasonably long for this stage of the project.

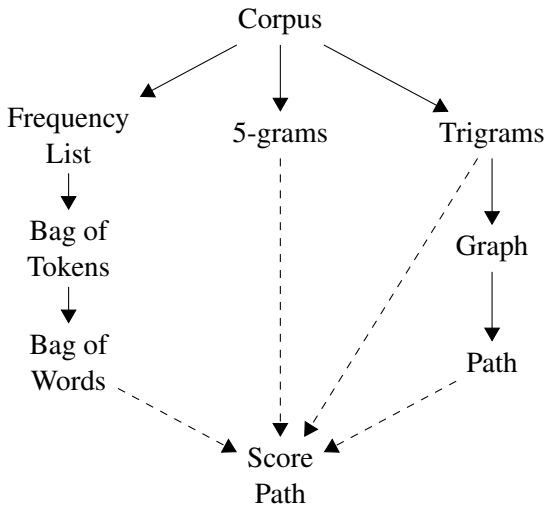

Figure 1: Simplified Overview of the Method

5.1 Create a "Similar" Bag of Tokens

First, create a corpus bag of words with document and frequency counts.

56	9281	the
56	6970	of
56	6840	,
56	4991	and
56	4676	.
	⋮	
1	1	medicine
1	1	chattel
1	1	blueprint

Second, cull the list. We chose to eliminate any words appearing in fewer than five of the speeches. Of the remaining words, we accepted any with a frequency over 1000, and randomly culled the rest with this test:

```
frequency > random.randint(1,1000)
```

which prefers words with higher frequency but may accept some with lower frequencies.

This culling left us with ninety-one words, which we took as the Bag of Tokens.

5.2 Zipfian Frequency Distribution

We made a bag of words by ordering the bag of ninety-one tokens by corpus frequency and giving the list a Zipfian frequency distribution, using `zipf = int(91/r)`, rounding down the quotient where r is the word's rank.

```
56  91  the
56  45  of
56  30  ,
56  22  and
56  18  .
      ⋮
 6   1  propose
 9   1  borne
 6   1  immediately
```

5.3 Trigrams

Our goal is to collect all trigrams we can link together to make a candidate document. Each useful trigram will contain only words in the bag of words, and it will be able to link to a preceding and following trigram. Two trigrams link if the last two words of the first match the first two words of the second.

We will use the corpus frequencies of the trigrams to score candidate documents. We preserve capitalization and punctuation for syntax.

Collect the (13695) corpus trigrams, and cull all trigrams that contain a word not in the bag of words (cut to 525). Then iteratively remove all trigrams that do not have both another trigram to precede and to follow (cut to 115).

```
25  the United States
22  , and the
17  of the United
13  . It is
12  of the people
12  fellow - citizens
 ⋮
```

5.4 Big Missing Step

We suspect we missed a big step here, that the culled set of trigrams does not contain some words in the bag of words. What we need is a set of trigrams that contains all of the words in the bag and only the words in the bag. If the trigrams do not contain almost all of the bag of words, there is no hope of constructing a good document from them. We need to iteratively cull both the bag of words and the trigrams until the two lists converge. Depending on the starting list of words, the sets may converge at two empty lists.

5.5 5-grams

The role of the 5-grams is to score the candidate documents according to the corpus frequency of the 5-grams they contain. Thus, we do not need to keep the 5-grams that contain words not in the bag of words, but the 5-grams do not need to link.

We start with 14763 5-grams and reduce it to 751.

```
4  of our fellow - citizens
3  of the United States .
3  the Constitution of the United
3  Constitution of the United States
3  of the United States ,
 ⋮
```

5.6 Graph of Trigrams

The trigrams form a directed graph, probably cyclic. Each trigram is a vertex, and there is a directed edge from trigrams A to B if the last two words in A match the first two words in B, that is, if B can follow A. Our task of creating a candidate document is analogous to taking a walk through the graph. We will limit the number of possibilities by only considering paths. [A *path* is a walk in which no node appears more than once (Wilson and Watkins, 1990).]

We used the python module `graph-tool` to create the graph and find all of the paths, each of which forms a candidate document. (Peixoto, 2014).

5.7 Candidate Scoring, with an Example

The highest-scoring candidate document in our sample run had fifty-nine words.

```
 , and the rights , and to the
rights and safety .  I have ,
in the power of the law , and
in its administration , and
with it , and that of which the
Constitution of the Constitution
, which is in the other , that
the safety with which we have not
, it is in
```

Those fifty-nine words came from these fifty-seven trigrams (with their corpus frequencies).

22 , and the
1 and the rights
1 the rights ,
2 rights , and
12 , and to
⋮
1 have not ,
1 not , it
4 , it is
2 it is in

Summing the corpus frequencies gives a trigram score of 150.

The candidate contains these corpus 5-grams, and earns a 5-gram score of 7, giving a total candidate score of 157.

1 in the power of the
2 of the law , and
1 in its administration , and
1 , and with it ,
1 the Constitution , which is
1 have not , it is

6 Ways to Improve this Approach

6.1 Measure Word Frequency Similarity

We hope later to implement the third scoring metric, the similarity between the word frequency in the candidate and the Zipfian word frequency of the bag of words.

6.2 Give Heavy Weight to the 5-gram Metric

The 5-gram metric gives the best insight into how well the candidate represents the corpus. Giving it two or three times the weight of the trigram metric may be appropriate.

6.3 Walk v/s Trail v/s Path

In the case of a directed graph like ours, a *walk* is a is a succession of edges connecting nodes. If all of the edges are different, the walk is a *trail*. If all of the nodes are different, then the trail is a *path*. If the first and last node are the same, then the path is a *cycle*. A cycle can become a path by removing the first or last edge. (Wilson and Watkins, 1990)

In mining the graph, our code (at this stage) only scores paths. It does look for cycles, but while it is possible that a cycle could score higher than a path, we have not seen it. Also, in the context of a speech, the first word should start with a capital letter and the last should be final punctuation in { '.', '?', '!' }. In such a document, the first and last trigrams would not overlap by two words; thus, a speech cannot be a cycle.

We limited our search to paths, rather than general walks, because in our previous attempts and in others' work we had seen the search get stuck in a cycle, generating an endless repetitive speech. It would score well, but not be good natural language generation.

We should consider adding some flexibility. Considering trails instead of paths would allow the same trigram to be used more than once, but not repeat cycles. Adding flexibility could exponentially increase compute time, however, so finding a balance would be a challenge.

7 Limitations of this Approach

7.1 No Gold Standard

We had no gold standard against which to measure our "best" documents; furthermore, such a gold standard would be problematic to construct and difficult to implement. Given a large corpus and a large bag of words (more than ten tokens), there would be many ways to make a "good" document. Which is "best"? How would one measure how close a given candidate is to the "best" document, and how would one automate the evaluation? The process would have so many layers of subjectivity that it would be difficult to create and implement.

7.2 Correct Path may not Exist

To construct a perfect document from a bag of tokens using this method, we would need a set of corpus trigrams that contain only, and all of, the words in the bag of tokens. Those trigrams would need to form a directed graph through which there is a path that hits each word in the bag of tokens exactly the number of times in the Zipfian frequency distribution in the bag of words.

While it may be possible to artificially construct a bag of tokens for which there would be such a set of corpus trigrams forming a graph with such a path, it is not likely that an arbitrary bag of tokens from the corpus would have such a path, even a path with most of the bag of words, especially if the corpus were small.

7.3 Compute Time

If the corpus were large, the corresponding trigram graph would also be large, and the cost of finding the longer paths is an $O(n!)$ problem. To know that we have found the "best" path, we would have

had to find and compared all of them. The path may exist, but we could not find in a reasonable amount of time.

If the graph is not connected, which is not unlikely, the problem can be broken down into searches in connected subgraphs. If the disconnected graph with n nodes is formed by connected subgraphs with p, q, and r vertices, then considering them separately reduces a problem of $O(n!)$ to $O(p! + q! + r!)$ with $n = p + q + r$.

8 Conclusion

We believe that our set of three metrics is novel and useful. Our approach of making the linked trigrams into a directed graph allows us to take advantage of existing optimized tools. We need to further explore whether it is possible to make a set of corpus trigrams that contains all of, and only, the words in a bag of tokens. It may be necessary to introduce some more degrees of freedom that will exponentially increase compute time. Finding an appropriate balance will be the challenge.

Acknowledgments

I would like to acknowledge the help of Dr. Aminul Islam for his suggestions and ideas from his lectures, and of my colleague Jennifer Mangum for teaching me Graph Theory.

Code

The code is available at https://github.com/bburkman/CSCE_561/tree/master/NLG. The main function is in `NLTK02_Start.py`.

References

Cristina Barros and Elena Lloret. 2015. Input seed features for guiding the generation process: A statistical approach for spanish. In *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)*, pages 9–17, Brighton.

Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media, Stebanstopol, CA.

Baptiste Fontaine. 2014. Trigrams-based text generation. https://github.com/bfontaine/trigrams. Accessed: 2018-12-03.

Kaori Kumagai, Ichiro Kobayashi, Daichi Mochihashi, Hideki Asoh, Tomoaki Nakamura, and Takayuki Nagai. 2016. Human-like natural language generation using monte carlo tree search. In *Proceedings of the INLK 2016 Workshop on Computational Creativity and Natural Language Generation*, pages 11–18, Edinburgh.

Martin Matulik. 2018. *Natural Language Generation from Structured Data*. Ph.D. thesis, Czech Technical University in Prague.

Tiago P. Peixoto. 2014. The graph-tool python library. *figshare*.

Pranjal Srivastava. 2018. How to create a poet/writer using deep learning (text generation using python)? https://www.analyticsvidhya.com/blog/2018/03/text-generation-using-python-nlp/. Accessed: 2018-12-02.

Robin J. Wilson and John J. Watkins. 1990. *Graphs: An Introductory Approach*. Wiley, New York.