Text and sentence generation using graph traversing techniques

Magnus Olsson
D03, Lund Institute of Technology, Sweden
d03mo@student.lth.se

Abstract

This paper describes a sentence generator system. It is a self-learning system with chat rooms as its primary source of input. The system uses graphs to store words and their relations, as well as some meta-data (frequencies, word attributes etc), and has a lot in common with a Markov chain. The goal is to generate grammatically correct Swedish sentences, based on the data collected from the chat rooms. The generator is completely stateless (sentences have no contextual dependencies).

As a project part in the Language Processing and Computational Linguistics course given at Lund Institute of Technology, I have implemented a number of possible improvements to the this system. I have also attempted to evaluate these changes, comparing the old system with the new.

1 Introduction

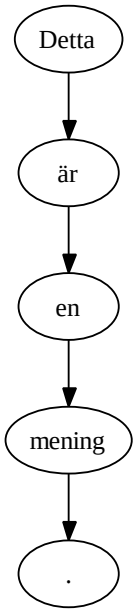
For a long time, all kinds of bots and AI (or at least what appears to be AI) has fascinated me. I started this project years ago, and since then it has gone through many development iterations. The goal has always been to generate text, with focus on the "correctness". I haven't bothered much with the actual content of the generated text, since the primary purpose for the system has always been to entertain and/or to serve as a "proof-of-concept" for my word graph idea. It's actually quite fun to let the bot learn from a channel, and then have it spew out a few sentences. The results can be very convincing.

Since the beginning, many things have changed. To my delight, most change seems to be to the better. In this paper you will find both the core idea, as well as some recently implemented improvements.

1.1 Theory

This section of the paper describes the core attributes of the word graph. As the bot processes incoming text from the chat rooms, each sentence is tokenized and each word is inserted into the graph. Relations to neighboring words are saved by putting edges between the word nodes. Each word is only stored once in the graph, duplicates are merged into a single node.

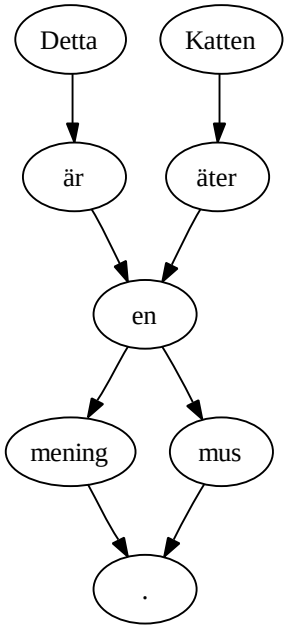
Given a Swedish sentence, "Detta är en mening." (*This is a sentence*), the system would decompose the text into the graph in figure 1.



Figur 1: One sentence word graph

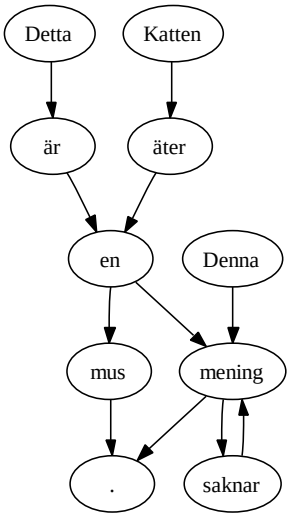
Adding another sentence, "Katten äter en mus." (*The cat eats a mouse*) adds the words "Katten",

“äter” and “mus” to the graph. Since duplicates are omitted, “äter” and “mus” is instead linked with the previously inserted “en” node.



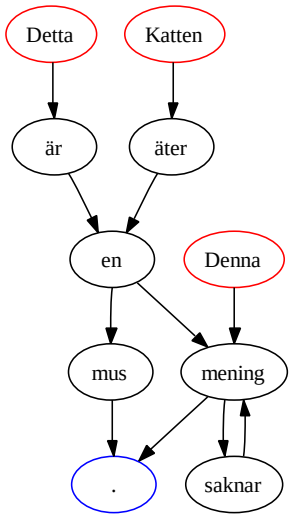
Figur 2: Two sentence word graph

Finally, if the sentence “Denna mening saknar mening.” (*This sentence lacks purpose*) is inserted, one may notice that a cyclic path is created between the nodes “saknar” and “mening”. This helps re-creating cyclic relations in generated text, but also introduces a problem when the generator spends too much time in the cycle.



Figur 3: Three sentence word graph

Each graph node also contains some meta-data, such as whether or not the word started (or ended) a sentence. In figure below, red outline means a start word while a blue outline means a stop word.



Figur 4: Red and Blue outline represent start and exit node respectively

This is used in the generation process, which can described in a few simple steps.

1. Pick an arbitrary start node (red nodes)
2. Randomly pick an outgoing edge and follow it. If there are none, go to last step.
3. Repeat from the second step until we encounter a stop node, then go to last step.
4. The sentence is created by adding all words together along the chosen path.

All randomness in the generation is uniform, this however changes with one of the improvements described later in this document.

2 Improvements

2.1 Part-of-speech tagging

A serious problem with the approach given in the theory section is that it does not respect the fact that some words in Swedish have an ambiguous part-of-speech tag. For example, consider the following three sentences:

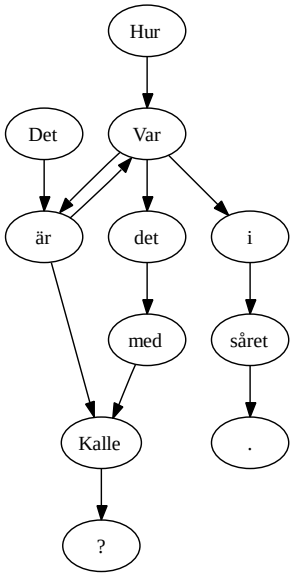
1. **Var** är Kalle? (Adverb)
2. Hur **var** det med Kalle? (Verb)
3. Det är **var** i såret. (Noun)

If these three sentences were to be fed into the graph, a single word node, “var”, would represent an adverb, a verb and a noun. This causes undesirable effects, since neighboring words in each of the three sentences expects “var” to be of the correct POS (part-of-speech) tag.

To solve this problem, a POS tag is inserted along with each word. In other words, a node is only considered a duplicate if both word and its POS tag matches. With the example sentences above, this would yield three different “var” nodes, each with a different POS tag (adverb, verb and noun). All edges are connected to their respective node.

One might argue that this reduces the number of possible new paths within the graph. This is true, but while it tends to reduce the number of paths, it also increases the quality of the generated sentences. Since far from all words in Swedish (or any language) is ambiguous, there are still a great number of new paths for the generator to utilize.

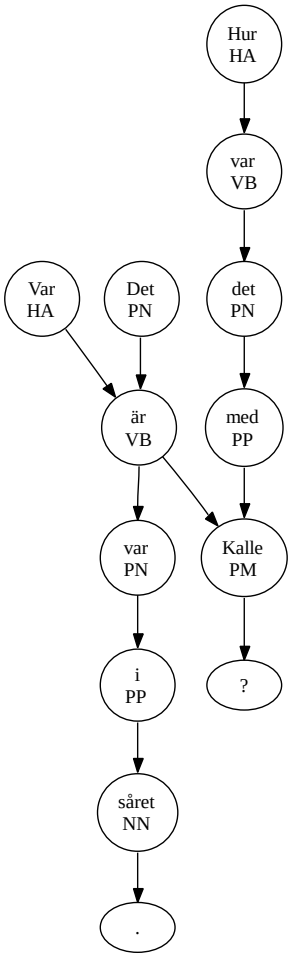
The following figure is an illustration of how the graph would look without POS tagging. Notice the edges density at the “var” node.



Figur 5: Word graph without part-of-speech tags

Figure 6 shows the the same graph, with POS-tags. The structure appears to be more organized, and while this graph still contains invalid paths, the trend is higher quality sentences using this improvement.

This technique creates dynamic part-of-speech patterns. A naive way to generate sentences is to hardcode the patterns into your software. For example, you could easily program software which



Figur 6: Word graph with part-of-speech tags

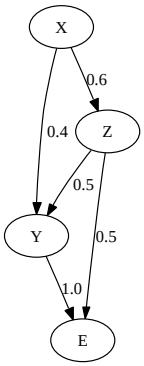
picks a sentence template from a list of predefined templates and fills its slots with appropriate words. A template could for example be “Pelle likes to eat [noun]”. Any noun would fit into this template, inserting “meatballs” would be just as correct as inserting “worms” or “trucks”. This approach however lacks the ability to learn new templates, it can at best only extend its list of words to insert at slots (nouns in this case).

Using a word graph with part-of-speech tagging, new templates (patterns) emerge naturally by simply traversing the graph. With a statistical approach, you can filter away noise patterns.

2.2 Markov chain adaptation

As the graph grows, noise becomes a problem. Very rare words have the same probability to occur in a generated sentence as the most common word. This happens because the original system uses a uniform distribution function to determine the next edge.

By adding weights to all edges, counting the number of times they have occurred, you can create a custom distribution function. With this approach, rare words still occur in generated sentence, but at a lower frequency. Just like it did in the training data. This creates a kind of threshold for noise, making the system favor words that are common.



Figur 7: Markov chain

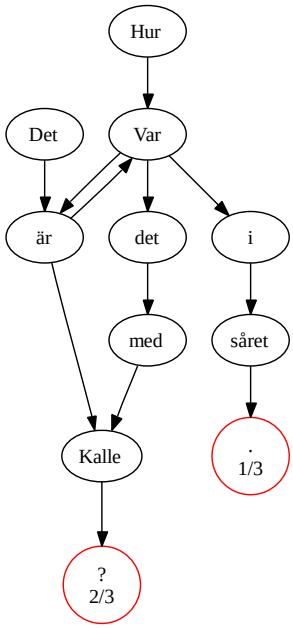
This model has a lot in common with Markov chains, where the model consists of a number of states and a number of transition probabilities.

2.3 Exit probabilities

In general, in Internet chat rooms, the language is quite relaxed. People do not always bother with ending each sentence with a punctuation mark. Compared to books, newspapers and other literature, the language differs a lot. This causes problems in the word graph, since it attempts to remember all words which can end a sentence. For small graphs, it works well by simply traversing the graph until we encounter an exit node. In large graphs however, many more words are tagged with the exit flag. The result is premature exits, we may only visit one (the start node is also an exit node) or two nodes before we exit.

To combat this issue, exit probabilities is added to each node, much like edge probability was added to rid the noise. With exit probabilities, common exit words (typically punctuation marks, . ? ! etc) have high exit probability. Other words, which may be marked as exit node only because of relaxed language is not favored, but still possible.

This figure illustrates the exit probability of three sentences. A question mark is a more favorable sentence terminator with a probability of approximately 67 percent, while the dot has an exit probability of approximately 33 percent.



Figur 8: Exit probability

3 Accuracy of part-of-speech taggers

It is crucial to use an accurate tagger, or else words may be incorrectly categorized and cause errors to propagate throughout the graph. Erroneous paths have great effect on the end results, since even slightly faulty segments of a sentence may have vast influence in how it is interpreted by the reader. Therefor, it is of utmost importance to have a high accuracy tagger backbone.

For Swedish, there are quite a few taggers to pick from. Most taggers are generic and can be trained on any language. To find an appropriate tagger, a number of taggers were trained using the Swedish SUC corpus and the results were to be compared. The candidates were,

- Granska, a hidden Markov model tagger.
- LingPipe POS Tagger, also a hidden Markov model tagger.
- Stanford POS Tagger, a log-linear tagger.

Unfortunately, due to technical difficulties the results of this comparison is inconclusive. Many hours were spent on trying to train models, and also to interface the taggers with the system. LingPipe was by far the easiest to use, and achieved a 94 percent accuracy in a 5 fold test on the test set. Due to some bugs in the Stanford tagger, it was unable to handle my training set. A fixed version is to appear on their website “in the near future”.

There were also difficulties training Granska, mostly because it lacks proper documentation and the tools to recreate the lexicon. It does however ship with a pre-compiled Swedish model, which works very well, but would not serve as an appropriate value to compare with. It also does tokenization on its own, which causes problems when you want to compare with other taggers. Granska is written in C++, and in order to use it from Java, one has to spawn an external process and capture its output. Java has facilities to do this, but due to some mysterious bugs (most likely connected to the IO stream handling in either Java or Granska) it would not work reliably. Despite all setbacks, a new Granska lexicon was eventually created, with the help and assistance from some of the Granska developers. Usage of this model, however, caused Granska to terminate with an segmentation fault.

4 Related work

Even though there are no real applications for this kind of system (nonsense generation), there are surprisingly many similar projects. Searching the Internet gives hundreds of different applications to generate papers and other texts. One project I found particularly interesting was SCIgen. SCIgen is a program that generates random Computer Science research papers, including graphs, figures, and citations. One of their generated papers, titled "Rooter: A Methodology for the Typical Unification of Access Points and Redundancy" got accepted to WMSCI 2005 (World Multi-Conference on Systemics, Cybernetics and Informatics), which is quite remarkable.

Eliza is a famous computer program, written by Joseph Weizenbaum. It parodied a Rogerian therapist, by rephrasing many of the patient's statements as questions and posing them to the patient. Just like the sentence generator, Eliza had no practical application, but it still impacted a number of early computer games; it influenced people write books; and it has received worldwide recognition. If all other fails, you may use it as entertainment!

5 Conclusions and future work

To determine whether the implemented changes in fact are improvements, it would be desirable to have a measurable quality value of each generated sentence. It is problematic to automatically measure the quality of generated sentences. A computer is not easily able to determine if the grammar is correct. In a large word graph, the number of possible paths (and sentences) are virtually unlimited. This makes manual quality grading impossible.

Still, manual grading on a small number of sentences is better than nothing. For this reason, a seven people group were asked to participate in a small survey. The test setup was 10 sentences from the old system and 10 sentences from the new (improved) system. These 20 sentences were mixed randomly, and each participant were asked to grade them (quality wise) on a scale 1 to 5. Average score for the old and the new system was then calculated. Here are the results,

Person	Old score	New score
1	3.2	**3.9**
2	2.9	**3.2**
3	3.0	3.0
4	**2.2**	2.1
5	1.7	**2.4**
6	**3.3**	3.0
7	3.0	**3.4**

A small test like this not enough to draw any conclusions, but it does however hint that the new system has at least equal (or slightly better) quality on its generated sentences.

Looking ahead, there are a number of possible interesting improvements to be implemented. In particular, it would be interesting to make the generator stateful, and to actually generate sentences on a topic of discussion. This would probably increase the credibility of the system. One possibility to achieve this is to implement a semantic graph on top of the word graph, linking words that are related to each other in a meaningful way. The Infomap Project at Stanford has already developed working semantic graphs which seems to work very well. It would allow the word graph to know that if the chat room conversation revolves around ammunition, a sentence containing a related word like "explosives" or "guns" would be more appropriate than a sentence about "breasts". This will most likely be implemented in the next generation of this system.

6 Acknowledgements

I would like to thank Pierre Nugues at LTH CS for his invaluable support and comments on my work throughout this project. I would also like to thank Viggo Kann and Jonas Sjöbergh at KTH NADA

for their efforts to help me getting the Granska tagger to run. Credits also goes to to William Morgan at Stanford NLP (POS tagger developer) for assisting with bug tracing using my Swedish model to find out what was wrong.

Finally, thanks to all who participated in the survey.

References

"Implementing an efficient part-of-speech tagger," Johan Carlberger and Viggo Kann, 2003 *Numerical Analysis and Computing Science, Royal Institute of Technology*.

"Granska tagger," http://www.csc.kth.se/tcs/humanlang/tools.html, *Human Language Technology Group at KTH CSC*.

"Stanford Log-linear Part-Of-Speech Tagger," http://nlp.stanford.edu/software/tagger.shtml," *The Stanford Natural Language Processing Group*.

"Markov models," http://en.wikipedia.org/w/index.php?title=Markov_chain, *Wikipedia, The Free Encyclopedia*.

"An Introduction to Language Processing with Perl and Prolog," Pierre Nugues, 2004 *Springer*.

"Semantic graphs (Infomap project)," http://infomap.stanford.edu/, *Computational Semantics Laboratory, Stanford university*.

"The Stockholm Umeå Corpus SUC," http://www.ling.su.se/staff/sofia/suc/suc.html, *Computational Linguistics at Stockholm University*.

"LingPipe" http://www.alias-i.com/lingpipe/, *Alias-i*.

"SCIgen - An Automatic CS Paper Generator" http://pdos.csail.mit.edu/scigen/, *PDOS research group at MIT CSAIL*.

"ELIZA," http://en.wikipedia.org/w/index.php?title=ELIZA, *Wikipedia, The Free Encyclopedia*.

"Wabby - Create semi-random sentences based upon a body of text," http://search.cpan.org/~poznick/Acme-Wabby-0.13/Wabby.pm, *Nathan Poznick*.

"Col's Random Sentence Generator," http://www.ast.cam.ac.uk/~cmf/generate/, *Colin Frayn*.

"Random sentence generation with n-grams," http://piece.stanford.edu/~brendano/rsg/, *Brendan O'Connor and Michael Bieniosek.*