POEM GENERATOR

DD2380 Artificial intelligence

Group 52

Jaldeep Acharya
jaldeep@kth.se

Paulos Kidane
paulosk@kth.se

Abstract

In this project a poem generator has been created using bigrams, a simple grammar model and Good-Turing smoothing technique. The poem fulfills the rhyming scheme "AABB". The chosen problem statement is "How human like poems can we generate using the chosen methods". Experiment on which grammar model is the best has been done. The conclusion

1. Introduction

Natural Language Processing (NLP) in Artificial intelligence is focused on developing systems that allow computers to communicate with people using everyday language. The system tries to understand the everyday language spoken or written by the humans and respond to them in the same way. It is required to build a robot or a machine to perform the tasks according to user instructions. Different branches of NLP are Natural Language Understanding (NLU) and Natural Language Generation (NLG).

1.1 Context and usability

The base branch for this project is NLG, where project context is within short text generation, or more specifically, poem generation. The strive will be to generate poetic texts that as much as possible can resemble the type of poem a human would write. There are a wide range of poem types with different characteristics. The characteristic focused on in this report are poems where pairs of sentences rhyme.

There are many areas in which a poem generator can be of use. An example of usage is for teaching in classes where poetic literature is studied. Students would be able to find clear distinctions between poets by having a larger set of poems that resembles the writing style of them. Poem enthusiasts could find this aspect of poem generation useful as well. In general, anyone interested in writing something poetic could use a poem generator. This could be either to create a complete poem, or to use it interactively to help with creativity.

1.2 Problem statement

The goal of the project is to create a poem generator that as much as possible resembles a poem written by a human. The poems will be texts of four lines where the lines follow the rhyming scheme *AABB*. The experiment discusses the suitability and extensibility of a poem generator that is based on *bigrams*, *grammar models* and *smoothing techniques* for *n-grams*. The bigrams and grammars will be based on a corpora containing 17 poems written in the rhyming scheme. The problem statement is to see how human-like the generated poems are.

1.3 Contribution

The current state of the project is in a development phase. This meaning that the ideal end-user application is not reached. Instead, the project can serve as a basis for a poem generator based on bigrams, POS-tag grammars and smoothing techniques and could be of much use for a developer intending to work on the topic.

For end-user purposes, the state of the project is viable as for *poem inspiration* only. The inspiration being the generated poems that can be corrected or changed in favour of the user. Similarly, inspiration can be found for rhyming words as a sub corpora is created for rhyming words.

2. Related Work

Poem generation has been researched before with varying techniques for targeting the problem. The main inspiration for this project has been[1]. Similarities between the projects is that a corpora of existing poems are used in conjunction with a n-gram (trigam) to predict words. Additionally, the approach for how to provide templates for words that are to be generated is similar. The topic of the cited project is concerned with *rhythmic poetry* where generated poems follows a pattern of stressed or unstressed syllables. An unsupervised learning process had been made in order to find syllables from words in existing poems. Generating poems will consist of finding appropriate words that would fit in a syllable pattern, in an iterative manner. The syllables would either be pre-defined or inputted for by a user. Similarly, this project applies the concept of unsupervised learning from a corpus of existing poems. Rhythmical properties of words and poems are not considered, but the syllable requirements for a generated word is corresponded with a grammatical tag that the word needs to have.

Defining grammars for poetry might be as complex as for natural languages as well. Inspiration for how to approach this problem is presented in[10]. The main influence is the usage of POS-tags for grammatical tokenization. By tokenizing a corpora of poems, sequences of grammatical tokens can be extracted. In conjunction with n-grams, the grammar corpora could be used to generate words following a probable grammar sequence.

For smoothing techniques a paper on comparisons of them[11] was used. The paper provides a comparison of many techniques, of which many are based on the *Good-Turing Smoothing*. The paper was used to give an overview of what kind of techniques were used and how they were used.

3. Method

The method chosen to create a poem generator is by using bigrams, a grammar model and Good-Turing smoothing. The bigrams are generated from a text corpus of poems, but instead of the standard way of reading the corpus from left to right, bigrams are created by reading from right to left. Bigrams have also been used to choose the next word that fits the poem generators *AABB* rhyming scheme. A strict set of predefined grammar rules have been used to formulate a grammar model and the Good-Turing smoothing has been used to give a little probability to unseen words, meaning bigram pair that didn't actually occur in the text corpus of poems.

3.1 Corpora

From a main corpus of 17 poems, the following sub corpora are generated:

- The main corpus with all words in the lines reversed
- A corpus with all rhyming word occurances
- A POS-tagged version of the reversed corpus
- POS-tag corpora for each POS-tag, containing all matching words.

3.2 Implementation

The poem generator uses two bigrams. The first bigram is used for generating sentences and the other one used for predicting the next word in the rhyming scheme *AABB*. To use a bigram for sentence generation that rhymes, the text corpus of poems is reversed line by line. An example to showcase this:

Normal: Wait for what this year brings
Reversed: brings year this what for wait

The reason for doing this is that the poem needs to rhyme and to make sure that the sentence ends with a word that fits the rhyming scheme, we need to generate the sentence backwards, starting by picking a word that fits the rhyming scheme. By reversing the text corpus, a bigram model can be created based on that corpus. Effectively, predicting words will be done in a previous first manner.

Bigrams are also used to select the next rhyming word. For this two bigrams are created separately. One which creates bigrams for all words that rhyme with each other, so either "AA" or "BB". The other bigram is used to get the transition from "AA" to "BB" in the poem, meaning words that are in the form "AB" in the text corpus of poems. A rhyming pair for the first two sentences is selected randomly. Given the rhyming pair, we have rhyming words A1 and A2. Given A2, B1 is selected from the second bigram, which is the word with the highest probability. Finally given B1, B2 is selected from the first bigram, which is the word with the highest probability. The selection of rhyming words in this manner is done until the grammar token for all the rhyming words A1, A2, B1 & B2, match the sentence end grammar token of the predefined grammar rules.

Figure 1 illustrates how the bigrams work to generate the poem. The 1 stands for the bigram which is created from the reversed corpus, giving us the probabilities of the words that should

occur previously to a given word, so $P(W_{n-1}| W_n)$. The 2 stands for the bigram one for the rhyming words which is used to predict word pairs that rhyme with each other, so “AA” or “BB”. The 3 stands for the bigram two for the rhyming words which is used to predict the transition from “AA” to “BB”, so “AB”. Given the figure, the poem would be generated by picking a word for W_5, then using bigrams we would predict all possibilities for W_4 choose the word with the highest probability using the bigram probability distribution matrix. So the word which has the highest probability for $P(W_{n-1}| W_n)$ is selected, then given W_4 predict W_3 in the same way as previously and so on until W_1 is reached. Then using W_5, W_{11} is predicted and even here we select the word with the highest probability using the bigram probability distribution matrix. Then sentence two is generated in the same way as sentence one. The rest of the two sentences are generated in a similar manner.

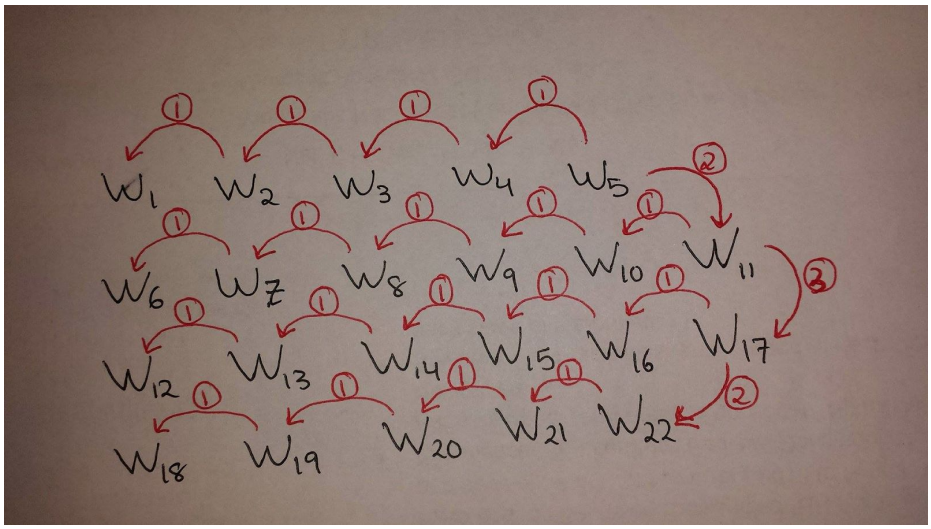

Figure 1. Use of bigrams to generate poem

Grammar model which has been used is a fairly simple one. 6 predefined grammar rules are used to generate a sentence. Each grammar rule has a unique grammar token for the last word in the sentence, so a grammar rule can be found for any rhyming word. Given the grammar model, the sentence is generated as described above. The difference now is that the previous word is selected from two criterias. The probability for $P(W_{n-1}| W_n)$ and the grammar token for word W_{n-1}, fits the grammar token G_{n-1}. The three highest probable words that fulfill this are selected and three different sentences are generated from there. This is done for each word that is predicted in the sentence, resulting in a lot of sentences generated. To evaluate the generated sentences and pick the best one a method was implemented that calculated for each word, the probability for that word of being between the two words in the sentence. This meant another bigram for words was generated, but this time on the main text corpus of poems instead of the reversed one. The probability of a sentence was calculated by summing the probability of word x_i by adding the probability of the word coming after word x_{i-1} and before word x_{i+1}, where i fulfills $1 <= i <$ sentence length. This was done for all the sentences generated, and the one with the highest score was chosen. This is illustrated in figure 2, where 1 is for $P(W_i|W_{i-1})$ and 2 is for $P(W_i|W_{i+1})$.

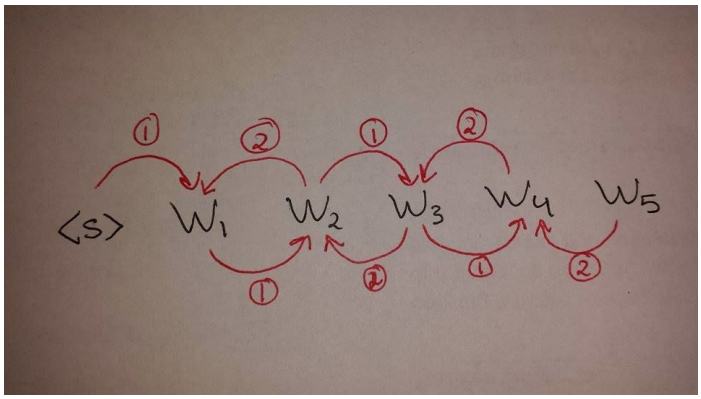

Figure 2. Method used for scoring a sentence

4. Experiment

This section will provide details on experiments that were conducted during the implementation of the project. A total of three experiments were conducted focusing on implementing grammar into the text generation.

The first experiment was to learn grammar from the text corpus of poems using bigrams. This meant POS-tagging the corpus and the creating bigrams. Hypothesis for this experiment was a high risk of repeated POS-tokens stuck in a loop. Since the grammar token with highest probability was always chosen, the expected result was in worst case a loop of grammar tokens, for example noun, verb, noun, verb, noun,.... The reason this experiment was conducted was to see if it was possible to learn grammar rules from a text corpus of poems using only bigrams, to generate custom grammar rules for poems with rhyming scheme "AABB" specifically.

The second experiment was to create sentences using existing grammar. This meant POS-tagging the corpus and then viewing each separate line in the POS-tagged corpus file as a separate grammar rule, and creating sentences for all the possible grammar rules. Hypothesis for this experiment was that it would generate a lot more sentences increasing the chances of one of the sentences being grammatically correct. The reason this experiment was conducted to see whether trying many different grammar resulted in a sentence that was both grammatically correct and somewhat sensible.

The third and last experiment was to set a couple of hard coded grammar rules, which in our experiment was 6 grammar rules, and create sentences considering more words than just the most probable on each step of the text generation. Hypothesis for this experiment was that we might get very limited sentences when it comes to the grammar, but still have a variety of sentences, since more words are considered at each step in text generation instead of always taking the most probable one. The experiment was conducted to see whether it was the variation of words that resulted in a better sentence rather than the variation grammar.

4.1 Experiment Setup

The first experiment was conducted by first creating bigrams of grammar tokens. This was done by part-of-speech tagging each word in the corpus file. Since the sentences are generated backwards to fulfill the rhyming criteria, we did the part-of-speech tagging backwards as well. Each word would also be written in a separate file for the specific grammar token, so after reading the whole corpus, one file for each grammar token would be available, containing all the words that were used as for example nouns in the corpus. These files were later used to determine the grammar token of a specific word. The grammar bigrams were then used when generating each sentence.

First the rhyming word would get selected, if it's the first word in the poem then selection would be random, otherwise it would be decided using the rhyming word bigram probability matrix. The grammar token would then be determined using the grammar token files and since a word could appear in several files, a method was written that returned a list sorted in descending order given a word, where the first element would be the most probable grammar

token for that word, the second element the second most probable grammar token for that word and so on. After determining grammar token for the rhyming word, grammar token for the next word is selected using the grammar bigram probability matrix. Using the word bigram probability matrix, a list of possible next words in descending order, the first element being the most probable word, is calculated. The list was used to determine the first most probable word that fits the grammar token. This was then continued until either a max limit of 10 words was reached or when encountering a word that was used as a sentence starter in the text corpus of poems and a the length of sentence was at least five words.
The second experiment was conducted by part-of-speech tagging the text corpus of poems, as done in experiment one. The difference was that instead of creating bigrams, each row in the POS-tagged corpus was seen as a separate grammar rule. A sentence for each grammar rule was then generated as in experiment one, where first the grammar token for rhyming word is determined, then for each grammar rule that ended its sentence with that grammar token were used to generate sentences. The difference from experiment one was that instead of predicting next grammar token in the sentence using grammar bigram, a set of predefined rules were used to generate sentence by selecting the most probable word that fulfilled the grammar. This generated a lot of sentences for a every sentence in the poem. The method described in the implementation section was used to grade the sentences and later pick the one with the highest score.

The third and final experiment was done by creating a set of predefined rules, which in the experiment was 6. The rhyming was then selected as before, then depending on the grammar token of the word, a grammar rule was selected. The difference this time from the previous experiments was to use x most probable words, instead of always selecting the most probable one when predicting the next word during text generation. For the experiment, the three most probable words which fitted the selected grammar rule were selected. This was then done for each step when generating a sentence. Viewing this as a tree, we can say that the number of words was our branching factor, which meant a lot of sentences were generated, since all combinations of the three most probable words were tried to generate a sentence. The method described in the implementation section was used to grade the sentences and later pick the one with the highest score.

4.2 Experiment Result

The result was in most cases a loop of grammar tokens, the most common one being verb, pronoun, verb, pronoun,...This was something that we were afraid of and were expecting. The reason behind is that since only bigrams are used and the most probable grammar token is always chosen, it could happen that pronoun is the most common token that usually comes before a verb and verb is the most common token that usually comes before a pronoun in the text corpus of poems, which means that a loop of these two grammar tokens would be created leaving us with a not so grammatically correct sentence. The result was still somewhat a little bit worse than our expectation, since we didn't expect the loops to be as common as they were. The reason was the use of bigrams to learn grammar from the text corpus of poems. A better result could have achieved if 3-grams or 4-grams were to be used instead of the bigrams to learn and predict the next grammar token, because we would take into account n-1 previous grammar tokens, rather than just the previous grammar token as in the bigram.

Result from the second experiment was a bit surprising. The expected result was that a lot more sentences would be generated, from which at least one would be grammatically correct,

and this was the case. The unpredicted part of the result was that all the sentences were generated using the exact same words, which meant that even though there were a lot more sentences to choose from, where some were more sensible, they all said the exact same thing since there was no difference in the choice of words. This was due to the fact that we had forgotten to take into account that the most probable word that fitted the grammar was always chosen, and since a word could be more than one kind of grammar token, it would lead to the same word always being chosen, no matter which grammar rule. This meant that even though the grammar was correct, the variation in sentences was zero and the words didn't fit well in the sentence making the sentence not so sensible. The hypothesis was partially correct, but it didn't account for the lack of variations in the sentences generated. A better approach would have been to have a better algorithm that chooses the next word more wisely than just taking the most probable word, leading to more variety of sentences. Another approach could have been to, like experiment three, try to create sentences with three most probable words at each step, although this would result in a very large number of sentences generated for each line in the poem, resulting in a long waiting time before the poem is generated.

The last experiment resulted in a lot more variations in sentences as expected. An unexpected outcome that we did not predict was the sentences being so grammatically correct even though only one rule was used to generate all the sentences. The reason being that even if only one grammar rule was used, it was still correct grammar which meant that all the sentence generated were grammatically correct, so all that was needed was to choose a sentence that had good choice of words that fitted that grammar as well as with the words that are exactly before and after, so that the sentence would be more sensible. A better function to rate the sentences could have helped even more to select a sentence that is a lot more human like. If we had looked at more words on each step of the sentence generation, then we would have generated a lot more sentences, resulting in more sentences to pick from.

5. Summary and Conclusions

Clear improvements in the text generator were seen after implementing each method step by step. When generating poems only using the bigrams, text was generated which didn't make sense grammatically. Neither did the sentence make any sense. An example is:

"Is regained when you were once spent,
When you drew me in between"

When then doing the experiments with different grammar techniques and finally choosing the one described in experiment three, a huge difference in sentence quality was noticed. The sentences picked were more grammatically correct and made a lot more sense since the best sentence from all the generated ones was chosen using a special method. An example is:

"Comes the morning everything is okay,
Exists only wants to stay
Loses its gloomy all go,
Worry dear santa i know"

The final step of adding Good-Turing smoothing technique improved the result a little bit, but nothing too significant. The reason being that only the 3 most probable words are chosen to generate sentence for which means that words with lower probability, including the ones smoothed in will in most cases not be chosen. Although there would be some cases where smoothed words will be chosen given that the original corpus didn't have many occurrences of a specific word meaning it won't have many suggestions for the next word, which is where the smoothing technique would come in and still give us three words to generate sentences for. An example is:

"Shine like worn out fashion,
Moments together were precious compassion
I'm dead at our consent,
Moments together were once spent"

From the progress that has been presented, it is evident that the sequential improvements in the techniques used yields better results in the generated poems. At this stage, single sentences can be created that make sense grammatically. The biggest problems are that two consecutive sentences not have much to do with eachother other than that they rhyme. Furthermore, the grammatical templates may not be optimal together. This is to be expected as the current state of the poem generator is line oriented.

Improvements can be made by having a larger corpus, which would be able to be more dynamic in choice of words. Changing from fixed grammars to more dynamic ones is a possibility as well. N-grams have been used for POS-tagged grammars[2], in order to dynamically create grammars that are probable together over longer word sequences.

In answering the problem statement; the answer is no. Single lines can be identical to human-written ones but not the complete poems. However, we believe that much can be improved in this matter. Considering the sequential progress that have been made, and the many more that are believed to be left, we conclude that there is a potential in creating a poem generator that can write poems resembling human-written ones.

6. Toolboxes

Stanford NLP - Core library: http://nlp.stanford.edu/

7. References

[1] Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation - ***E.Green***. *Haverford College;* ***T.Bodrumlu****, Univ. South California;* ***K.Knigh****,Univ. South California*

[2]POS-Tag based Poetry Generation with WordNet- ***Manex Agirrezabal***, University of basque country; ***Mans Hulden***, University of Helsinki

[3] An Empirical Study of Smoothing Techniques for Language Modeling - ***Stanley F.Chen****, Harvard University;* ***Joshua Goodman****, Harvard University*