CS 155 Miniproject 2: JustTryIt Team Report

Kangchen Bai, Pengchuan Zhang, Yerong Li

1 Overview

2 Data Manipulation

The basic strategy in our model training is to train multiple models for different part of the poem. Therefore, we have the following pre-processing data manipulation.

Grouping Shakespear's sonnets enjoy the clear rhyme scheme *abab cdcd efef gg*. Moreover, lines with different rhymes have quite different sentence structure. For example, the first and third lines *aa* have quite different sentence structure from the last two lines *gg*. Based on this obersevation, we decide to train 6 models for these 6 parts, namely *a, b, c, d, e, f, g*. Therefore, we first group all the lines in the same part of the poems and get six corpuses, namely `groupA`, `groupB`, `groupC`, `groupD`, `groupE`, `groupF`, `groupG`.

Punctuations Shakespear's sonnets contain various punctuations (e.g., " ,", " '", " -"). We delete all punctuations except " -", and thus the words "Feed'st" and "'This" become "Feedst" and "This". For words with hyphen " -", we manually delete it or replace it with empty space " ". There are in total 83 hyphens in `Shakespear.txt` and it is very easy to deal with hyphens manually. After this stage, we have six corpuses and every corpus contains hundreds of lines without any punctuations.

Tokenization We tokenize the words as features and use the method `text.CountVectorizer` from `sklearn.feature_extraction` to preprocess every corpus. In simple, `CountVectorizer` lowercases all the words and builds a dictionary between the words and the natural numbers $\mathbb{N}$. The output of this tokenization step is six corpuses with sequences of natural numbers. These six corpuses will be the input of our model-training algorithms, e.g., HHM and 2nd-order Markov model.

To achieve better poem-generating performance, we generate each line in the reverse direction with pre-sampled rhyming ending words. We keep generating lines until we get a line with exactly 10 syllables. In order to achieve these additional goals, we have the following pre-processing data manipulation.

Generating rhyming dictionary We use the *NLTK* package and the RhymeBrain website `http://rhymebrain.com/en` to build a dictionary for rhyming words. For more details, see Section **??**.

Counting syllables in each word We use the *NLTK* package, the *PyHyphen* package and our own-written function `count_syllables()` to count the number of syllables in each word. These three methods have their own advantages and disadvantages, and we combine them to get the most accurate syllables-counting. For more details, see Section **??**.

3 Unsupervised Learning

We worked on Hidden Markov Model and Markov Model in this project.

Hidden Markov Model

In training HMM, we tried on several number of hidden state in our model and chose the numjber of state with the highest emission probabilities as the favorate model in the project. To be specific, we tried on models with 5, 10, 20 , 40, 80, 100, 500, 1000. We are working on model with 1000 hidden states because there are around 3000 words in Shakespeare's poetry set.

Here is a figure illustrating the performance of HMM with different state. Here the vertical axis is ... NEED SOME TEXT

Markov Model

1st order Markov Chain Model

The relationship between HMM and original Markov Chain Model We see in history that Hidden Markov Model is necessary when the number of observation states (number of words in Shakespeare's Sonnets, in this case) is unavoidable large. Besides this, we need Hidden Markov Model because we are supposed to get some intuitions in the grouping of words and hidden states are bringing us information. But in the case that the number of different words is affordably large (there are around 3000 words in this case), Markov Chain proves a more sophisticated model.

With this consideration and with the purpose of generating more reasonable verse set, we also worked on **Markov Chain model** in this project. In the basic (1st) Markov Chain Model the joint probability is given by

$$p(x_{1:M}) = p(x_1)p(x_2|x_1)p(x_3|x_2)...p(x_M|x_{M-1}) = p(x_1)\prod_{m=2}^{M} p(x_i|x_{i-1}) \tag{1}$$

But when we first get our trial on this **first order Markov Chain Model**, it does not give us perspective result, because this is extremely similar to what we have done in our **(1st order) Hidden Markov Model**, as what we have stated above, instead of tokenized the words into phrases, we tried the second order model instead, for the simple reason that this will do the tokenization automatically and is much more subjective in tokenization.

2nd order Markov Chain Model

In the **second order Markov Chain Model**, the assumption on the transition probability is:

$$p(x_m|x_{1:m-1}) = p(x_m|x_{m-1}, x_{m-1}) \tag{2}$$

So, different from the first order model, the joint probability in the 2nd order Markov Model gives:

$$p(x_{1:M}) = p(x_1, x_2)p(x_3|x_2, x_1)p(x_4|x_3, x_2)...p(x_M|x_{M-1}, x_{M-2}) = p(x_1, x_2)\prod_{m=3}^{M} p(x_m|x_{m-1}, x_{m-2})) \tag{3}$$

What should be mentioned is that we do counting and normalilzation for computing the piror probabilities $p(x_1x_2)$ and trains on the transition probabilities $p(x_m|x_{m-1})$. Since we have around three thousand words

in Shakespear's Sonnet, we the number of parameters (for $p(x_1x_2)$ and $p(x_m|x_{m-1})$) is not substantially large, so the running time for 2nd order Markov Chain Model in affordable.

Notes on some trials and improvements

Here are some of the trials we have worked on in data manipulation and training:

- NEED SOME TEXT

4 Visualization and Interpretation for HMM

5 Poetry Generation

We present results from the models we worked on in this project. As stated above, we do counting in the poetry generation to make sure that each line in our poem consists of 10 syallables.That is to say, we actually generation our poetry line by line, at each position of line, we repeated generate lines until we get a 10-syllable line: we only took our pick of lines with 10 syllables. In counting the syllables, we use dictionary from *NLTK* and package *PyHyphen* to break words into syllables, we did not truncate lines during the counting, so each line is supposed to end up in the *END* state (this is the same for both Hidden Markov Model and Hidden Markov Model). We only took our pick at sentence level.

1st Hidden Markov Model
2nd order Markov Chain Model

6 Additional researches that we have worked on

7 A reference for our document

8 Conclusion