Machine Learning & Data Mining
Miniproject 2

Caltech CS/CNS/EE 155
Released February 17^{th}, 2017

---

1 Introduction

- **Group Members**
  Gavy Aggarwal & Abirami Kurinchi-Vendhan

- **Division of Labour**
  Gavy worked on data input/output, processing and parsing the input poems, training the Hidden Markov Model, and writing the word selection algorithm.

  Abirami worked on setting up the development environment, designing the poetry generation algorithm, and enforcing the syllable count and rhyme scheme of our sonnet.

- **Source Code**
  We have open sourced the code we used to generate our sonnet. You can find it at:
  https://github.com/abiramikv/shmmakespeare

2 Tokenizing

- **Initial Approach**

  We started by taking a primitive approach and read each file of sonnets line by line (filtering out lines that had less than 30 characters in it to account for the headings). Then, we took every unique word and mapped it to an integer that was used as a token to train the model. Thus, each line of words corresponded to an individual input sequence where the length of the sequence was equal to the number of words in the line.

- **Case and Punctuation**

  After implementing our initial approach, we discovered that there were some duplicated words, and words like ”you” and ”You” and ”you?” created multiple tokens for what essentially represented the same semantic concept. Thus, we modified our initial approach by formatting each word by converting it to lower case and stripping punctuation from either end of the word when creating our mapping from word to integer token.

- **Sequencing per Sentence**

  After we had implemented our poetry generation algorithm, and we were trying to improve the accuracy of our grammar and semantic meaning, we decided to modify how we tokenized the input. We looked at the raw data and saw that the same logical structure extended across multiple lines, so tokenizing on a line by line basis could reduce the accuracy of our model. We decided to change our tokenization strategy and form sequences of inputs based on sentences that we could parse across multiple lines. We did this by combining multiple lines together and splitting them based on punctuation such as colons, periods, question marks, exclamation marks, and reaching the end of a sonnet.

- **Tokenizing every Syllable**

  Instead of tokenizing every word, we tried tokenizing every syllable and assigning either a "stressed" state or an "unstressed" state to it. However, we dropped this method as when we generated words, we came across several fictional words that were constructed from combining various syllables and didn't make sense in our output poem.

3 Algorithm

- **Packages**

  To train our model, we used the hmmlearn Python library. Specifically, we used the MultinomialHMM model, and trained using input data with variable lengths.

- **Parameters**

  While training our Hidden Markov Model, we focused primarily on two factors: the number of hidden states to include and the number of iterations to run our algorithm for. Initially, we started at 10 hidden states and 10 iterations, but after experimenting with both factors, we decided to use 10 hidden states and 3 iterations in our final model.

- **Effect on Sonnets**

  Looking at the effect of the number of iterations on the sonnets, we saw that past two iterations, there was no improvement in the quality of our sonnets. After performing some further investigation, we saw that the model always converged after two iterations and running it for 10 iterations took longer to train without adding to the quality of the sonnets, so we chose to run the algorithm for only 3 iterations for good measure.

  We saw a similar outcome for the number of hidden states. When using 5 or less states, we could notice a significant reduction in the quality of the sonnet as the words had poorer grammatical structure and made less sense. When producing sonnets with 20 or more hidden states, the quality seemed roughly the same, but the training time increased significantly. Since the quality of the sonnets could not be numerically quantified, we decided to keep 10 hidden states for good measure as a tradeoff between quality and runtime.

4 Poetry Generation

- **One of Our Generated Poems:**

  To powers to away you ever are,
  Sake skill back lovely one of adorn with.
  Fearless more be thou day of have set or
  Cruelty scope in the can life since this smith

Is the which of every by self clock.
Against choose if come whom stupid mortal,
Begin do men life of degree lease mock.
Augurs when critic I which doth do will,
As guide in o thou but me doth so day.
But termed not a by spirit compile do,
O you be his thy which so sight dear say,
By so haste be then be what so your due?
Thine where when shall fate sullied what that for,
I flow to sun heat to true dress true more.

- shmmakespeare.py

- **Generation Algorithm:**
  - **Line Count:** The first constraint that we imposed on our sonnet was a length of 14 lines. This was relatively easy to implement as we generated the sonnets line by line and simply stopped after the fourteenth line was completed.
  - **Syllable Count:** Next, we wanted to ensure that each line of the poem consisted of ten syllables. To determine the number of syllables in a word, we wrote an algorithm that utilizes the Carnegie Mellon Dictionary of words and pronunciations from the NLTK database. The algorithm first checks if the word in question is in this dictionary. If so, it counts the number of phonemes with numeric stressor indicators to determine the number of syllables. If the word cannot be found in this dictionary, which was often the case due to Shakespeare's predilection to make up his own and remove letters as he saw fit, another algorithm that counts the number of vowels (including 'y') preceded by consonants. The algorithm handles special cases of words that end with 'e' and words that have no vowels. Since this algorithm does not handle all special cases properly, we only used it as a back up to the NLTK method, which we found to be more accurate because the phonemes were manually assigned. Since our lines were generated word by word, we checked the number of syllables after the addition of each word to the line. Once we hit ten syllables exactly, we determined that the line was finished. If we exceeded ten syllables, we used back-propagation to reset the sequence to previous state we could build off of (see section below on Back-propagation).
  - **Form:** Our final constraints were related to form. We noticed that in most sonnets, the final couplet is indented. For that reason, we added a few spaces at the beginning of those lines. We decided to also capitalize the first word in each line to follow Shakespeare's form. To address punctuation, we found that random assignment of punctuation worked poorly. We also found that we were unable to train the model to be able to predict punctuation much better than random assignment, which detracted more from our poem than contributed to it. As a result, we decided to skip punctuation as part of our post-processing and ultimately added it in manually as we saw appropriate.

- **Word Generation:** To generate the words themselves, the starting probabilities were used to randomly pick the first token. The observation matrix and transition matrix were used to generate all subsequent tokens based off each previous token randomly. The sequence of tokens was then detokenized to form the sonnet.
- **Back-propagation:** If we reached the end of a line unable to meet the syllabic and rhyme constraints for the line, we removed the last three words and regenerated more until we reached a sequence that met our specifications. Initially, we removed just the last word and resampled it given the previous state, but we found that it often resulted in selecting a word that didn't really fit, yet was forced to be in that position for the sake of rhyming or being a set number of syllables. By replacing the last three words, we found that the sequence felt more natural while still retaining the proper rhyme scheme.

Also see Additional Goals for further improvements that were involved in generating the poem above.

- **Overall Analysis:**
The poems generated strictly abide by the ten syllable count, the sonnet length, and Shakespearean rhyme scheme. The areas where our sonnets deteriorated from the typical form were grammar, meter, and theme. We were able to notice some strings of words that made grammatical sense in our poems, but there were some errors that were littered throughout. Similarly, there were parts of our poems that followed iambic pentameter, but the trends weren't consistent enough to guarantee that the poem followed the desired meter. Unfortunately, our poems were most severely lacking in theme. There were not really any coherent messages in our poems. While there were individualized sections that made sense, they didn't work together to tell a story. Additionally, they did not conform to the Shakespearean plot structure where a problem is introduced in the first three quatrains, and then resolved in the final couplet. These deficiencies were expected as we did not explicitly account for them in our poetry generation and largely relied on the HMM model to develop these trends. As a result, these characteristics could not be guaranteed in our poems.

5 Visualization and Interpretation

- **State Associations**
For our top 10 states, here are the top 10 words that are associated with each hidden state:

State 1: and in to the my thy me when thou your

State 2: the and i in that thy of love with my

State 3: and to my that of the is in with be

State 4: the that in i and to thou when with all

State 5: to my and not the of that all so thee

State 6: i to the my that of in and a which

State 7: the of my and but i thou to a her

State 8: of i my the but thee which be with for

State 9: of and to in with i but thy her the

State 10: and that to the in thy love me it i

- **State Analysis**
  Looking at the top words associated with each state, we were expecting to find some type of logical association between the states. For example, each state could represent a different part of speech and the top 10 words in each state would share the same part of speech. However, since we used a purely unsupervised model and let the Hidden Markov Model algorithm determine the starting probabilities, transition probabilities, and emission probabilities, the end result included several of the same prepositions and determinates as the top words for each of the 10 states. Another possibility was that states grouped words of a similar connotation or syllable count, but after looking at the output, that is clearly not the case. While we weren't able to find a clear semantic pattern distinguishing all the states, we suspect that the different states correspond to different parts of the sentence (beginning, ending, middle, etc.), and the top words happen to be prepositions and determinates simply because they appear with the highest frequency in Shakespeare and Spenser's works (and in the English language).

  We tried generating a sonnet and outputting the states instead of the words, and got this:

  5 8 3 6 5 7 3 6 8
  8 7 0 6 9 2 7 9
  9 8 2 5 7 0 1 2 3 8
  5 3 5 3 6 1 0 6
  9 8 3 8 5 8 3
  8 4 2 4 8
  9 0 1 0 3 4 4 4
  8 7 3 6 1 5 2 5 6
  1 6 8 0 5 9
  7 2 5 3 7 7 1 1 6
  1 3 5 3 7 4 5 2 4
  0 8 8 8 2 1 5 6 4
  7 9 6 7 7 3 4 6 4
  9 6 6 6 1 8 9 4

  Here, we observe that most lines end with the states 4, 6, and 8 while most lines start with 1, 5, 7, and 9. This is further evidence that the states correspond to different positions in a sentence/line.

- **Visual Representation**

cloudy when mirth her my o love most thy
self which eyelids full unquiet the love ill
saw cold dear so with worth if soul sport i
to lose phoebus but leap bright and we will
within and put not who not was nothing
joy for all of of and foul entice got
to each be i fears heart thine self hearing
hast to ne far doth face his unto not
now than you sins tie thou her; with help you
of of let i the me steal should here my
whilst my or but his janus with the view
wild all temptation all this heart to eye
exchequer to vain but in means in as
to if but stay by then who field too says

Consider the coloration of this sonnet based on the state. The 10 states were divided into three groups: beginning, middle, and end. Each group was then assigned a color: blue, green, and red, respectively. Note from observing our sonnet, that the blue states are found mostly at the beginning of each line, the green states are found mostly in the middle, and the red states are near the end of the line. Thus, our various states correspond to different line positions in our model. Obviously, this trend isn't perfectly represented in the sonnet given above. This is because we sampled the states randomly, so there is the chance of random error causing an ending start to appear at the beginning of the line and vice versa. Also, backtracking and resampling several times to find a word that rhymed or had the correct syllable count also created a probabilistic bias that reduced the accuracy of the placement of our states.

6 Additional Goals

- **Rhyme:**
We decided that implementing the *abab cdcd efef gg* rhyme scheme would be play the largest role in making the sonnet more reminiscent of Shakespeare's work. To implement this, we utilized the NLTK cmudict API. We determined if two words rhymed by comparing their last two phonemes. If they matched in any combinations of pronunciations of the words, we decided they rhymed. We found this generated good results nearly all of the time. Decreasing this bar to one phoneme increased the ease of finding words that "rhymed," but generally returned weaker rhyming pairs. When we increased this bar to three phonemes, finding words that rhymed became much harder and increased our poem generation time. For this reason, we settled on a limit of two. Additionally, we analyzed

all pronunciations of the words since it was hard for us to determine the context for pronunciation programatically. Also, since some of Shakespeare's words were not defined in the cmudict library, we decided to avoid using them at the ends of sentences so that the algorithm would not be stuck on trying to find rhyming pairs for these words. This was imposed on every line of the poem. Though this limited our usage of Shakespeare's vocabulary, we decided that the tradeoff was worth it since we could successfully enforce a rhyme scheme. Further, we added a precaution to prevent the algorithm from rhyming words with themselves since it made the poems seem less legitimate in practice. The aactual rhyming constraint was enforced on the second line of each of the rhyming pairs. If the algorithm produced an ending word that didn't rhyme, it reverted to the back-propagation method discussed in the Poetry Generation section. Adding the rhyming scheme improved the sound of the poem but because certain words were forced to rhyme, they sometimes did not make sense in the context of their surrounding words. We decided that the rhyme scheme improved the sonnets overall since they didn't make perfect grammatical and comprehensible sense to begin with.

- **Additional Texts:**
  We also trained our HMM on additional texts to improve our model. Including the Spenser's Amoretti provided us with more training data that was somewhat similar to Shakespeare's method of writing, since he was one of his contemporaries. We noted that the model was more capable of formulating strings of words that made sense once we expanded our training set to include Spenser. We tried to incorporate other English texts from different time periods and others to strengthen this effect, namely *Pride and Prejudice* by Jane Austen. We noticed that these texts detracted from Shakespeare's style rather than match it so we decided to limit ourselves to using the Spenser texts.

- **Meter:**
  We attempted to account for meter in our poem by using the stressor indicator on the phonemes generated by the NLTK cmudict library by analyzing all the pronunciations of every word. We were able to semi-successfully incorporate iambic pentameter in this manner. However, we quickly realized that enforcing stressed/unstressed syllables severely impacted the grammatical and logical sense of the poems. Consequently, we decided remove meter from our manual constraints and rely on the HMM to capture it in the model.

7 Conclusion

Given that we had written a program that could generate poems from scratch, we were impressed and satisfied by the results. However, after observing several minor grammatical mistakes and seeing the lack of patterns during state analysis, we wish that we made a more robust model by using semi-supervised learning or by using a technique called "anchor states" that disallows certain predefined words to appear from a different state, causing the patterns in the states to be more pronounced.

While these changes would help improve grammatical accuracy, we also hoped that our poems would make semantic sense. We believe, however, that the Hidden Markov Model we were using (at least

with 10 states) was not enough to capture the semantic meaning, and we'd need to use more sophisticated techniques to create a Shakespearean sonnet that had thematic elements and told a story.