Machine Learning & Data Mining
Miniproject 2

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

---

1 Introduction

- **Group members:** Enrico Borba, Claire Goeckner-Wald
- **Team name:** Papa Mart's Mini Gary - The Comeback
- **Division of labour:** Enrico Borba: Programming, ideas, report visualization. Claire Goeckner-Wald: Programming, ideas, report assembly.

2 Pre-processing

- **Handling the dataset**
  - **Quatrains versus couplets:** Initially, we thought that we should train two different models, one on lines from the quatrains, and one on lines from the couplets. This way, we could more accurately capture the "shift in tone" that Shakespeare often performs during his sonnets. However, since we decided to 'force' rhyming using a rhyming dictionary garnered from the dataset, we ended up combining quatrains and couplets into one model.
  - **Punctuation:** We stripped the following punctuation from the sonnet data, including the parentheses: $, . :; !?()$
  - **Tokenization:** We tokenized on the words in the sonnets, essentially giving the HMM each line of every sonnet to train on. We also lower cased all of the sonnets.
- **The dictionary**
  - **Structure:** We used a standard python dictionary to assign each word or part-of-sentence a unique number. The dictionary was double-encoded, meaning that there was an entry for each word $w \to x$ and each number $x \to w$.
  - **Reason for use:** The HMM would take only numbers, not words or characters. We had to set all word to lowercase first, to avoid using some word "The" in the middle of a sentence, when we would rather have "the".
  - **Unexpected trouble:** The use of the dictionary is also one reason why we decided to treat lines from quatrains and couplets equally. Initially, we had one large dictionary that covered all words Shakespeare used in the dataset. As expected, some words were used only in the couplets, or only in the quatrains. However, we ran into errors when training two separate Hidden Markov Models, most likely because the model expected that if we gave it words (represented by numbers) $\{3, 15, 23, 14, 194\}$, that there would be $(1 - 194)$ states available. However, this was not the case.

3 Unsupervised Learning

- **HMM**
  - **Naive poem generation:** We used the HW5 solutions for the Baum-Welch implementation.
  - **Packages:** We used native python for tokenization into words. We then used, nltk to count syllables, along with a heuristic for words that nltk did not previously know.
  - **Number of hidden states:** 6 states seemed to yield (loosely) the most coherent poems. Attempting with greater than 10 states seemed to produce erratice sentences (ones containing multiples participles and uses of "I"). Training with 2 states yielded a very simplistic model, essentially selecting words at random with probabilities proportional to their occurence.

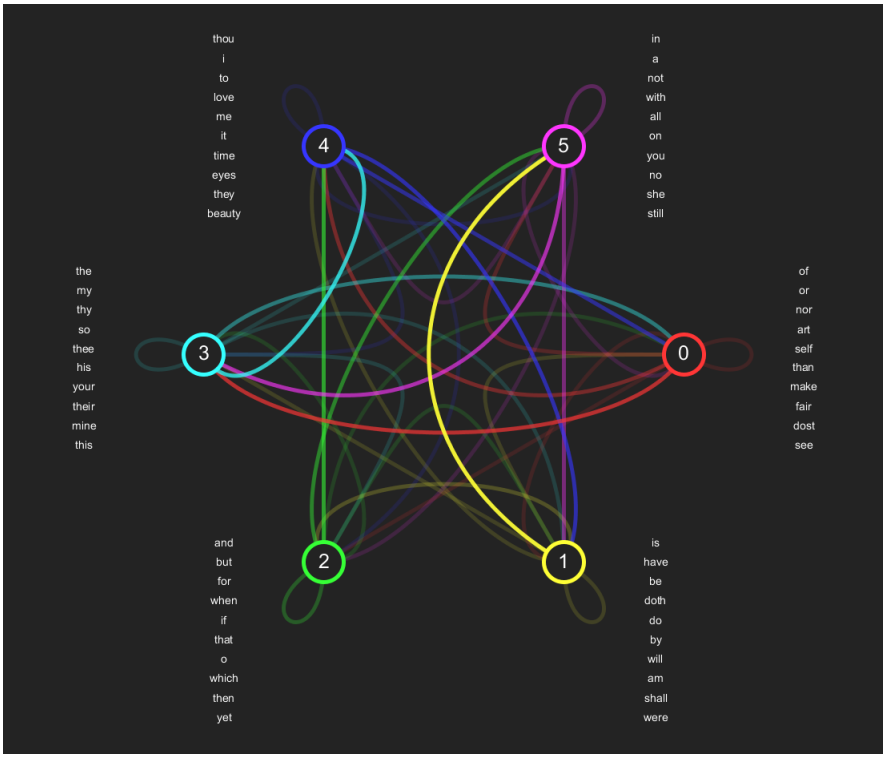
4 Visualization & Interpretation

- **Interpretations**

  - **Analyzing the model:** We tokenized the sonnets by words, removing punctuation. Because of this, the HMM can only determine patterns between words. Thus, the 6 HMM states have some complex pattern between the words, unlikely to be too related to the English grammar.

  - **Imagery:** The below image shows the 6 states and their transitions. Each state is colored to show directed transitions. That is, since state 1 is yellow, all yellow transitions come from state 1. More transparent transitions show lesser probabilities. So, the nearly opaque transition from state 1 to state 5 is of comparitively high probability. Also, the transitions are drawn in order of their probabilities. So, if some transition a appears above transition b, the transition a has higher probability than transition b.

    Furthermore, we have the top 10 words for each state represented next to the states. However, we disallow repititions of words in the image. If a word appears next to a state, then that state was the state that was most likely to emmit it.



– **State Analysis:** We notice that state 1 has a high collection of verbs. This continues as well: checking the top 20 words for state 1 yields "are, may, was, should, say, did, must, know, might, away", which is further composed mostly of verbs.

  State 1 has the highest probability of transitioning to state 5, which contains mainly prepositions and some nouns (which correctly follow from verbs).

  State 5 has the highest probability of transitioning to state 3, which has several possesive nouns. Inspecting the top 20 words of state 3 yields more possesive nouns such as "her, thine, our", and also some adjectives.

  State 3 then has the highest probability of transitioning to state 4, which contains even more nouns, as the sentence at this point, if began at state 3 (a likely verb), should reach the noun the verb is modifying.

  State 4 then has similar probabilities of transitioning to states 1 and 0, which allow for more compound phrases. Since the HMM mainly trained on single lines of the sonnets, taking more than 9 transitions almost guarantees a grammerless phrase. The average number of words per line in all of the sonnets is just over 8, thus the HMM has difficulty outputing coherent long phrases.

5 Poetry Generation

- **Algorithm**
  - **Two example non-naive sonnets:**

```
1  She have not eyes bright be that tiger's moan
2  Every must change touches thou moan bed
3  But the gold shook all so let and upon
4  Everywhere to although stain full unbred
5  To and i'll so shows care more releasing
6  Be it thee thou fore when your which rehearse
7  They the rage these life of mine possessing
8  Painting therefore thine night a for inhearse
9  His thy spurring prove so from are compiled
10 In leisure in and seeing receives cry
11 To touches strangle thy the to me filed
12 Him my that saith they have my jollity
13     Remain lawful hast I say 'will' annoy
14     A under is his name strong thinks fled joy

1  If breath to were book you due look rotten
2  Line his in or you steeled my thee shown
3  If wilful-slow grace me when forgotten
4  Of every pen loving summer own
5  Me going deserve with filching not guard
6  Is dear of love in paying live it leaves
7  Blame ever worst o what gazing since ward
8  Your for candles distraction there be sheaves
9  Excel heart others small invention light
10 The with write way thy so health ignorance
11 Then which thus do beweep need'st them aright
12 And an far day back my in count advance
13     All and face lo an sweets sight over-plus
14     My respect triumph it uncertain thus
```

  - **Algorithm:** We first randomly pick 7 pairs of rhyming words. Then, since we trained the HMM in reverse, we feed each of these 14 words in the HMM to yield 14 lines of 10 syllables. In order to ensure the proper number of syllables, we pick a random number of words (in accordance to the distribution of the number of words in each line), and query the HMM to emmit a sentence of that length until it has exactly 10 syllables. If this does not occur within 1000 attempts, we generate a new random sentence length, and attempt the process again. We then reverse these emmissions and assemble them in the "ababcdcdefefgg" manner.

- **Quality of generated poems**
  - **Rhythmn:** We did not encorporate stressed and unstressed syllables.
  - **Syllables:** In order to have 10 syllables, per line, we ran a script to gather data on the average number of words per line in Shakespeare's sonnet. The distribution was approximately ([5 words] * 2 , [6 words] * 15 , [7 words] * 41 , [8 words] * 70 , [9 words] * 63 , [10 words] * 25). We used this distribution data to pick a random number of words, generated that many words (minus one, since we picked the ending word to force rhyming), then would continously generate new lines until we had a line that was 10 syllables using our counting algorithm. To count the number of syllables in a line, we mostly used the NLTK. However, there were some words used by Shakespeare that were not included in the NLTK. So, we supplemented this process with a less-accurate but still quite useful syllable-counting algorithm that would give its best guess wherever NTLK failed.
  - **Grammar:** The grammar was perhaps the most difficult part of the poems. On occasion, we would get two words in a row, such as "I I" or "the the". More commonly, we would see two parts-of-sentence that did not make sense together, such as "a the", "an a", or "thee thou".
  - **Punctuation:** Since we stripped all punctuation during pre-processing, there was minimal punctuation afterwards. Most notably, we would occasionally see the word 'Will', with the apostrophes. We could have easily added ending punctuation, since Shakespeare has a fairly consistent style of , , , :, , , :, , , :, . as line endings. However, we chose not to, for aesthetic reasons.

6 Additional Goals

- **Improving poem quality**
  - **Rhyme and the rhyming dictionary:** We forced rhyming by creating a rhyming dictionary of all of Shakespeare's ending words. We iterated through the sonnets, creating pairs of rhyming words using the known rhyme scheme "ababcdcdefefgg", then used a clumping algorithm to create super-pairs. For example, $(head, bed)$ and $(head, dead)$ would combine to $(head, bed, dead)$, then search again for additional matches. So, when building a line, we would pick a random rhyming word, build the line 'backwards', then pick another rhyming word from the rhyming set (sometimes the same word if the size of the set was one), then repeat. Then, once we had 7 rhyming lines, we simply organized them into the "ababcdcdefefgg" rhyme scheme.
  - **Syllables:** We properly handled words with hyphens and apostrophies by splitting words on those delimiters.
  - **Punctuation:** We readded capitalization and punctuation at as a post-processing step.