Miniproject 2: Generating Shakespearean Sonnets using First-Order Hidden Markov Models

Daniel Gu
Victor Han
Eve (Yi) Yuan

1 Overview

In this project, we wanted to learn a generative model which can write Shakespearean sonnets. Using Shakespeare's 154 sonnets and Spenser's sonnets from his sonnet cycle *Amoretti* as unlabeled training data, we tried to train a first-order hidden Markov model (HMM) to generate sonnets similar to ones Shakespeare could have written.

A lot of the challenge in this miniproject is that first-order HMMs seem to be too weak to be able to perform this learning task. Sonnets have structure at varying levels of granularity, such as the iambic pentameter, which means that the previous word affects the current word, the syllable count, in which all of the previous words in a line affect the current word, and the rhyme, which is a "long-distance" interaction among line endings. And this is just the *syntactic* structure of the sonnet: the stanzas and couplet of a Shakespearean sonnet also need to be endowed with meaning. So a lot of the challenge was augmenting or clever tweaking our data to try to capture this structure using a first-order HMM.

More concretely, a challenge we ran into was the amount of time needed to train a first-order HMM. An off-the-self implementation such as the one by the Natural Language Toolkit takes a lot of time (a few seconds per iteration for 10 hidden states, significantly more at higher hidden states), and our implementation took similar (and usually longer) times to finish training the HMM. So it took a lot of time to check for errors in the EM algorithm implementation, and after it was debugged, a lot of time to look for a good number of hidden states for our HMM. We were also constrained in time since looking at higher numbers of hidden states took too long. To help alleviate the problem of too much training time, we tried training on subsets of the poems at a time, which did help in the debugging process. However, even with using only a few poems, using large numbers of hidden states still took their toll. Another problem appeared to be that Shakespeare does not repeat words very often. Thus, it seemed like we did not have very much training data on most individual words, and thus their roles were probably difficult for the HMM to figure out. A final problem was that it was difficult to judge the quality of the poems generated. So, what works and what doesn't work was difficult to determine.

Daniel wrote some parsing code as well as used an off-the-shelf implementation of unsupervised training for HMMs (nltk.tag.hmm) to look for the best number of hidden states, and wrote the code to create rhyming poems. Victor wrote most of the unsupervised training code, wrote a little preprocessing code, and also looked for the best number of hidden states. Eve wrote some parsing code, most of the visualization code, and most of the poem generation code.

2 Data Manipulation/Preprocessing

Given text files of Shakespeare and Spenser's sonnets, we parsed them into individual poems, which was represented as a list of its words in order. This seemed best since lines or bigger are too coarse a level of granularity (we don't want to just recite resorted Shakespeare lines), using syllables would cause us to generate imagined words, and parsing words could allow our model to capture parts of the structure such as the meter. We did end up writing a syllable parser anyways, but using it was soon abandoned.

In tokenizing as words, we also made a few other decisions. We kept hyphenated words together as a single token. After all, there must be a reason they are connected, and keeping them together may make our poem more meaningful/consistent. We kept punctuation with the words they come after. We wanted to keep these kinds of voice in the poem, and figuring out where to place them in poem generation would be difficult. Parentheses, however, were removed in preprocessing because the HMM cannot model opening and closing parentheses by itself. Initially, we did not take out parentheses and soon discovered that we got random closing parentheses and opening parentheses with no clear place to put closing counterparts when generating poems.

Other than straight up processing and tokenizing into words, we also did processing based on the number of unique words a poem had. In the function, parseTokLimMin() in parse.py, given a number of poems to search through from Shakespeare and Spenser and a total number n of poems to find, we tokenize only the n poems whose combination yields the fewest number of unique tokens. That is, if a combination of three particular poems uses 176 unique tokens in total, and all other combinations of three poems uses 177+ unique tokens, we choose the set of three poems with 176 unique tokens. The reason for this preprocessing is two-fold. The first is that if we have fewer unique tokens, then we have more data on each individual token, so that the HMM can better train on it. The second reason is that we are very constrained on training time, and having fewer tokens in general reduces the time needed to train.

3 Learning Algorithm

The major algorithm we had to implement was the Baum-Welch algorithm for unsupervised training of first-order HMMs. The Baum-Welch algorithm is a variant of the expectation-maximization algorithm for HMMs. Given training data in the form of unlabeled sequences, we first randomly initialize our transition matrix A and our observation matrix O, which together completely characterizes an HMM. We then run the expectation step, in which we calculate maximum-likelihood marginal probabilities using the forwards-backwards algorithm as a subroutine. In particular, the forwards algorithm calculates given A and O using a Viterbi-like dynamic programming algorithm the values

$$\alpha_Z(i) = \Pr[x^{1:i}, y^i = Z | A, O]$$

that is, the probability of observing the length-i prefix of a sequence x with the ith hidden state y^i being state Z. Similarly, the backwards algorithm is also a DP algorithm which calculates

$$\beta_Z^(i) = \Pr[x^{i:M}, y^i = Z | A, O]$$

that is, the probability of observing the length-$M - i$ suffix of sequence x with the ith hidden state being Z. Once we have these, we can calculate the marginal probabilities according to

$$\Pr[y^i = Z | x] = \frac{\alpha_Z(i)\beta_Z(i)}{\sum_{Z'} \alpha_{Z'}(i)\beta_{Z'}(i)}$$

$$\Pr[y^i = b, y^{i-1} = a|x] = \frac{\alpha_a(i-1)A_{a,b}O_{x^i,b}\beta_b(i)}{\sum_{a',b'}\alpha_{a'}(i-1)A_{a',b'}O_{x^i,b'}\beta_{b'}(i)}$$

Once we have these marginal probabilities, we can then perform a maximization step, in which we update A and O according to these marginals via

$$A_{a,b} = \frac{\sum_{j=1}^{N}\sum_{i=0}^{M_j}\Pr[y_j^i = b, y_j^{i+1} = a]}{\sum_{j=1}^{N}\sum_{i=0}^{M_j}\Pr[y_j^i = b]}$$

$$O_{w,z} = \frac{\sum_{j=1}^{N}\sum_{i=0}^{M_j}1_{[x_j^i=w]}\cdot\Pr[y_j^i = z]}{\sum_{j=1}^{N}\sum_{i=0}^{M_j}\Pr[y_j^i = z]}$$

We can then repeat this until it converges. For our implementation, we stopped when the sum of the Frobenius norms of the difference between the old and new A matrices and of the difference between the old and new O matrices was very small compared to the first algorithm iteration difference.

4 Naive Poem Generation

Using the trained A and O matrices, we can then generate a sequence in accordance with our trained HMM by randomly choosing a start state (according to the distribution described by the HMM), and then randomly choosing state transitions and token emissions in accordance with the probabilities described by the HMM. To enforce the 10-syllable per line structure, we used a tool to count the number of syllables and stop until we had 10 (although, due to the limitations of the tool, some lines may end up with 11 syllables).

We trained our naive HMM model using a variety of subsets of poems and number of hidden states. Models trained with a high number of hidden states and small number of poems naturally had good performance, since there is a high probability of matching Shakespeare's words or phrases:

When thou o present with I not think too, whom
Me, now I do, doing thee vantage, do
Do I not my self I in still with sighs
Them, moan? What merit do I not thy mind, those
Those that whom that thou my best thy part I
I to whom bending that against my self with
With thee vantage, double-vantage me.
Me. Such is my self, to my self me, art
Art present blind. With sighs and they with thee. Glory:
Or heart am still with whom time thou shalt on
On me do I not my self with thee vantage,
Partake? Double-vantage me. Such is
Is my love, to my self I do, doing
Doing thee partake? Do I not my self

This poem was generated on the 3 Shakespeare sonnets with the least unique tokens with the number of hidden states equal to the number of unique tokens (176). It still doesn't make semantic sense, but a lot of phrases are natural English phrases.

We also did tests with all of the poems on a small number of hidden states. 12 hidden states seemed to generate good poems, compared to the number of hidden states around it:

Have sings hath thou heretic, they best all love, is
Such nought the old, all the for when name have,
On thy way, with admitted muse with wiry
The burthen in it know more though alone the
They like among the when after new worth then
So common thy quietus take, them if steal
Do I th' which day, within therefore desire
Shall may not to subjects why all and of
From repair it may nor faith be came sweets
Divine, summer's (my blind. Cross. Maturity,
Friends in they injurious hardest the
Believe hymns blooms nor I her longer hast cupid
Deceived. Strange: let thy sick wane and all my
Within, dear o I yet eyes for what compare

Although the model is clearly learning something (it seems like it understands conjunctions to some degree, for example) the bulk of the poem is all totally nonsense. This model was trained only on Shakespeare's poems. If we train with Spenser's poem as well, we get a poem like

Excuse and pride, insight, is (being weave.
They with too fair it to ye of one deny,
Breaches is wrack. Nor at make treading been to
Down hell bring for thy runs my heart: to base
True garden come mind: and countenance habitation
With in new beauty, to what assurance:
Sake: was that not so crew: where with angel's
Left pine, to some the wander only do my
Not your the thou bend, heresy minutes last their
Alone, shinedst whom it is my (my laugh by not,
And would and tread death's a fair from feeble
Sickness pine will the blossom of it eyes
Leach turns contrained do they weave. And alone, they
Skill: this to absence on spotless defeat,

The poems generated are not clearly better than those generated with Shakespeare's sonnets alone. It is possible to discern a shift in style, though; the model trained only on Shakespeare sounds (to the authors, at least) more like Shakespeare than ones trained on both Shakespeare's and Spenser's sonnets, in which a bit of Spenser's style can be seen. Also, note that the model is not particularly good at replicating the meters.

Due to the time needed to train unsupervised HMMs, exploration on other types of models was not possible.

5 Better Poem Generation

To create rhyming poems, we first used Shakespeare's and Spenser's sonnets to create a "rhyming dictionary", which internally looks like a list of lists, with each inner list represents a rhyme equivalence class; that is, every word in the list rhymes with every other word in the list. We then seeded the end of each line

of our sonnet with matching pairs (if our HMM training set included these tokens) according to the rhyme scheme (Shakespeare's more relaxed $abab\ cdcd\ efef\ gg$ scheme), and then generated each line backwards in accordance with our trained HMM.

An example of a rhyming poem is

She time's her long with old body so elsewhere,
Suit painted nor me as so a to foregone,
Name do cherish: likewise whose her your near.
Purchase love's of lusty predict as her worth. Gone.
Heavy me and what deep to you taken. Poor.
With greater by thy outward heavenly to glad,
Past him her another eternal being store.
Hath o banks, of make like loves, to me sad.
Splendour golden bower in my tongue, flatter read.
With for mutual done confounds self he canopy,
Of strength them herself her so the basest plead:
Hold, to both sang. By before? Eternity,
Night seen, for stealth hell, the glorious raised.
Be. Then think your fortune's await tormenteth, praised:

The rhymes are generated well. The poems themselves do not seem to be worse due to the generation method, but since they were mostly nonsense to begin with, it's hard to tell if there's improvement.

6 Model Selection

We didn't do anything too sophisticated to do model selection; we trained our HMMs with different numbers of hidden states, and then examined the poems and chose the number of hidden states which made the poems we thought were qualitatively best. Due to the time required to do unsupervised HMM training with large numbers of hidden states, when using all of the poems, we limited our search to numbers of hidden states below 17.

In the end, using hidden states between 8 and 17 didn't really seem to change much. The poems were all gibberish. They could somewhat follow proper grammar, but meaning-wise, there was nothing. Meter seemed to be hit or miss. Due to the very random nature of the resulting poems, though, if the poem generation algorithm was run enough times, surely a great, creative poem could pop out.

In order to train on a large number of hidden states, we also used the preprocessing method described earlier to choose just a few poems (3-6 usually) and train on them with numbers of hidden states on the order of the number of unique tokens. One example is where we trained 3 poems with a total of 176 unique tokens using 176 hidden states. The results definitely made more sense and had better meter. However, the cost was that the generated poems had more phrases taken directly from the poems they were trained on. With the number of hidden states equal to the number of poems, however, this was not too much of a problem, as the size of the directly taken phrases seemed to cap at about 4 words. Most of the in the generated poem were still unique to the poem, though, and if phrases were taken, the most common size by far was 2 word phrases.

So, in the end, we mostly experimented on two extremes. One where there was a lot of data and few hidden states, and another where there was not much data and many hidden states. Time constraints would not allow us to test out the case of a lot of data and many hidden states. For these two extremes that we were able to examine though, the pros and cons can clearly be seen. If we want more "creativity," we should go for more data and less states (8-12 seemed fine). If we want to be more Shakespeare-like, we should go for more hidden states.

In the end, we decided to go with a poem generated from the more hidden states model, as it generated lines that more often made sense than the less states model.

7 Visualization

We trained essentially two kinds of HMMs: Part of speech modeling HMM and word modeling HMM. For the former, we trained with numbers of hidden states close to the number of parts of speech (8) in English. For the latter, we used large numbers of hidden states, equal to some fraction of the number of unique words present in the poems we used to train the HMM.

State	Noun	Verb	Article	Adjective	Adverb
0	0.41	0.17	0.11	0.22	0.09
1	0.14	0.07	0.15	0.12	0.09
2	0.30	0.23	0.01	0.13	0.12
3	0.35	0.15	0.00	0.16	0.22
4	0.39	0.24	0.00	0.18	0.24
5	0.61	0.36	0.00	0.22	0.07
6	0.38	0.27	0.00	0.26	0.17
7	0.15	0.10	0.00	0.15	0.13

Table 1: Probability of emitting each parts of speech per state

Poem Line	State Sequence
Early nor releasing: with my shame or foul	1,5,7,3,0,5,7,0
My fresher will so blind, matter bequest be	0,5,0,4,3,2,4,3
Unused swear that side, of the days. And the	3,6,1,5,7,1,5,7,1

Table 2: Poem lines and the state transition sequence that produced them

For the 8 state HMM, we expected to see some states have high probability of emitting one part of speech and a low probability of emitting all others. However, as demonstrated by Table 1, we saw some states have high probability emitting multiple parts of speech and some states have low probability of emitting all parts of speech with respect to the other states. This may be the result of the imperfect methods we use to visualize parts of speech. We determined that there is no good way of determining the part of speech of a single word without any context. The method we chose to approximate this is to check words against a large database of already tagged words (specifically nltk's wordnet corpus). However, the corpus

we use only has 60k nouns and probably does not include a lot of the older terms used in these poems. Interestingly, state 1 in the table has a much higher probability of emitting articles compared to other states, and lower probability of emitting other parts of speech compared to other states. This may indicate that state 1 represents articles to some extent. This is supported in Table 2. We see in row three, in the line "Unused swear..." both times the article "the" appears in the line, it is emitted by state 1. The state isn't a perfect article representation, however, as it also emits some non article words, such as "early" in row 1 of Table 2. State 5 has the largest probability of emitting nouns out of the states, and we see in Table 2 that the tree nouns that appear ("shame", "side", "days") are all emitted by state 5. This indicates that state 5 may represent nouns to some extent. A possible explanation for why we emit other parts of speech from these states is that there are ambiguities to words when we can't examine them in context. For example, in the line "Unused swear..." swear can either be a noun or a verb. In the context of the line, it's more clear that it's a noun, but our visualization does not examine the poem for context (as it often doesn't make much sense grammatically). This would also explain why many states have high probabilities of emitting nouns, verbs, and adjectives, as there are many words that could be any combination of those three parts of speech without context clues. So, states such as state 0 may represent these ambiguous words, with certain weights towards noun, verb, and adjective. There are also parts of speech that we weren't able to visualize, the biggest of which is prepositions. Fully visualizing all parts of speech would give us a fuller, more accurate picture of these states.

For the word modeling HMM, many of the states had 0 or 100 percent emission probability of a noun, verb, or article. This makes sense if each state represents a particular word or groups of words. In one such HMM, where we trained on 5 poems with 176 hidden states (half the number of unique words between the poems), we found that the words emitted by state 132 were: "long", "wary", "chary", "conquest", "would", "smother", "either", "farther." This is a group of 4 negatively toned words ("wary", "chary", "conquest", "smother") and 4 neutral words ("long", "either", "would", "farther"). This could indicate that state 132 represents a particular group of negative words (with some neutral noise words). Similarly, we see that state 116 emits the following words: "thou", "kind", "heart", "obsequious", "says", "eye's". Once again, we have 3 toned words (positive this time): "kind", "heart", "obsequious" and 3 neutral words: "says", "eye's", "thou". This indicates that state 132 is a group of positive words. In this HMM, we generally see only one kind of toned word (positive or negative) along with a bunch of neutral words being emitted from each state. This indicates that in this formulation, the states most likely represent groups of similarly toned words. The inclusion of neutral words in the state emission groups may be explained by the fact that there are far more neutral words than there are toned words.

The many state HMM case seems to perform better in poem generation than the 8 state case, as it generates poems that have better grammatical structure. However, for this formulation, with the high number of states used, there is danger in overfitting our training data. This would result in our generated poems simply parroting phrases from Shakespeare or Spenser. In Figure 1, about half of the states only have one state with a higher than 0.1 transition probability. This could mean that this formulation of an HMM is closer to a Markov chain, which only links words that were linked together in the original text. However, we also have about half of the states showing more than 1 possible option for transitioning. Upon visual inspection, we see that our poems are mostly unique poems without any phrases used in our source poems longer than 2 words. To be extra sure, we ran our poems through plagiarism checkers, and found no plagiarism present.

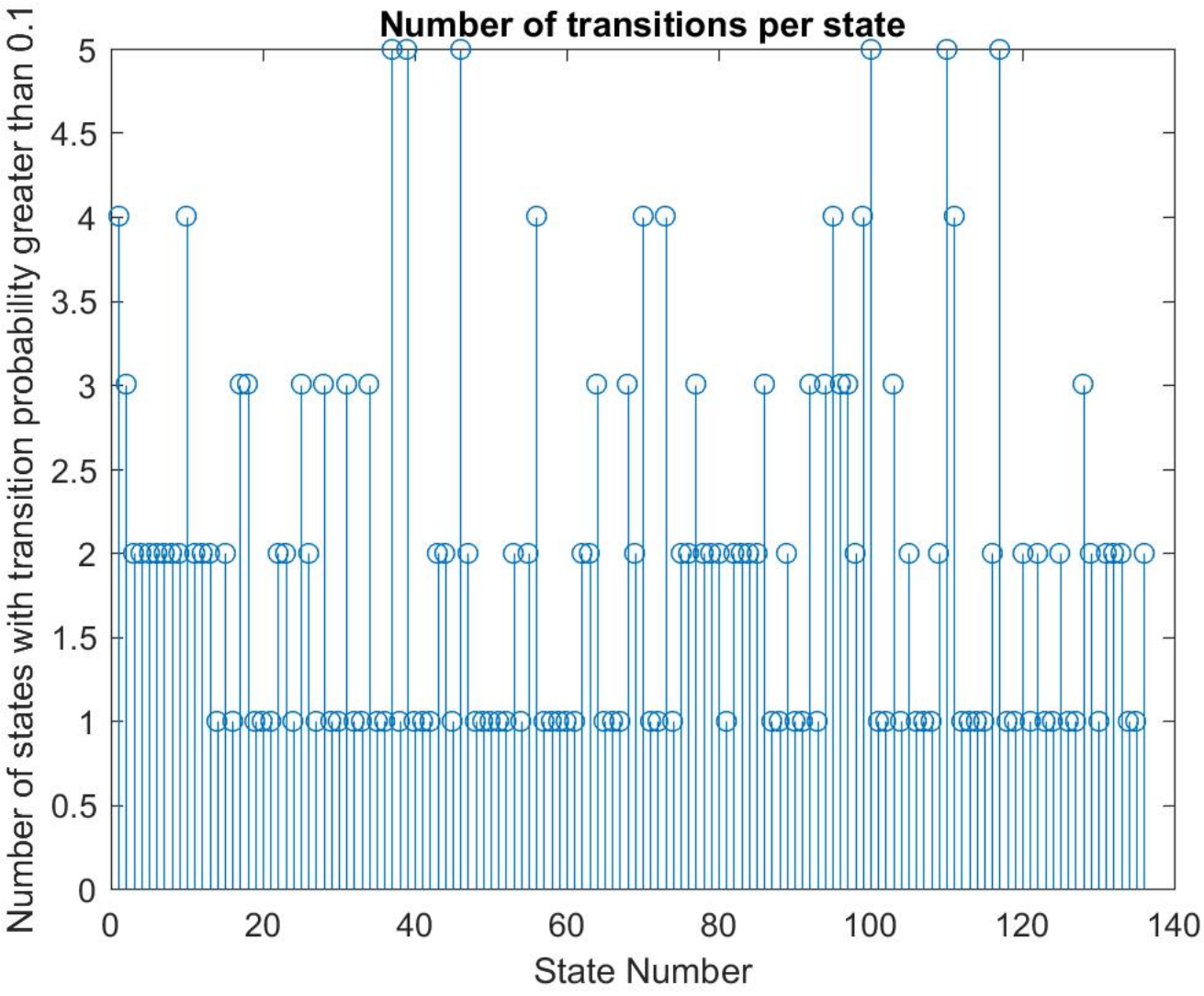

Figure 1: State number vs the number of states with a transition probability greater than 0.1 from that state

8 Conclusion

We trained a variety of hidden Markov models to generate Shakespearean sonnets. Constraints of the sonnet form we could achieve were limiting each line to 10 syllables, and introducing end rhymes. We were not able to replicate the meter, or create semantically meaningful poems. As stated in the overview, it seems that first-order hidden Markov models are too weak to model the complicated interactions that make up a sonnet (without even considering the difficulty of making poems with meaning). Possible directions to improve our model could be more detailed search over the space of hidden states and tokenizations to find better HMMs, research into clever tactics to allow HMMs to capture this structure, and switching to more sophisticated models (such as more complicated graphical models) to try to better capture the sonnet structure.