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

Daniel Gu
Victor Han
Eve 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. [TODO: discuss other challenges here.]

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. [TODO: add what Eve and Victor did.]

2 Data Manipulation

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. We also did parsing of poems into sequences of syllables. [TODO: discuss more preprocessing]. This seemed best since lines are too coarse a level of granularity, and parsing words could allow our model to capture parts of the structure such as the meter. [TODO: other stuff?]

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 change in log probability does not change significantly. [TODO: discuss numerical stability issues and other stuff].

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). [TODO: discuss more.]

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 according to the rhyme scheme (Shakespeare's more relaxed $abab\ cdcd\ efef\ gg$ scheme), and then generated each line backwards in accordance with out trained HMM.

4 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, we limited our search to numbers of hidden states below 17. [TODO: verify and discuss more if necessary].

5 Visualization

State	Noun	Verb	Article
0	0.53	0.30	0.00
1	0.12	0.07	0.00
2	0.48	0.36	0.10
3	0.23	0.07	0.19
4	0.51	0.31	0.00
5	0.10	0.05	0.00
6	0.17	0.13	0.00
7	0.28	0.14	0.00

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. For the 8 state HMM, we expected to see some states have high probability of emitting nouns, others verbs and particles. However, as demonstrated by the chart above, we saw multiple states have higher probabilities of emitting all three part of speech types compared to other states. This may be the result of the imperfect methods we use to visualize nouns and verbs however. We determined that there is no good way of determining if a single word, without any context, is a noun or a verb. The method we use 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 3 in the chart has a much higher probability of emitting articles compared to other states, and lower probability of emitting nouns and verbs compared to other states. This may indicate that state 3 represents particles.

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.

6 Conclusion

[TODO: discuss poem quality and stuff]