- Group members: Stavya Arora, Isha Goswami, Ivy Brainard
- Team Name: SII
- Colab link:
  - https://colab.research.google.com/drive/1bpBN_vSwoDR0QZtUpTA8UWEUcqPvbX0n?usp=sharing
  - https://colab.research.google.com/drive/1yOHBX5KqwldsqdmXPd7d3fgeB-A5aGMU?usp=sharing
- Piazza link: https://piazza.com/class/m4xf8vvl4206zc/post/290
- Division of labor: Each of the group members contributed equally to this project. We each tried creating various visualizations and analyzed key concepts and findings together.

1 Introduction

This report explores the use of hidden markov models (HMMs) to generate Shakespearean-style sonnets by learning from a dataset of all 154 of Shakespeare's sonnets. HMMs are particularly powerful statistical models that can represent Markov processes between unobservable (sometimes referred to as hidden) states.

Through extensive pre-processing, we stored all of the poetic features of the data set, including word sequencing, punctuation, and line breaks, later using this for generating emissions after training the HMM to produce accurate poems. By training the HMM on this data set, we aimed to produce coherent, structured poetic text that adheres to traditional constraints such as syllable counts, line structure, and rhythmic flow.

We also trained a recurrent neural network (RNN), specifically a character-based long short-term memory (LSTM) model on this Shakespearean sonnet data as another mode of poem generation. The LSTM model expands on the RNN by learning long-term dependencies, useful for text generation. Training on characters proved to be much more difficult in terms of generating a coherent poem, as discussed further in section 6.

Beyond basic sequence generation, we introduced constraints to enforce poetic structure, particularly in rhyme and meter. To capture the rhythm of Shakespearean verse, we implemented syllable constraints, seeded lines to follow the 'abab cdcd efef gg' rhyme structure, and adjusted the HMM's state transitions to enforce iambic pentameter. This refinement helped generate poetry with a more natural, rhythmic quality. This report demonstrates the drawbacks as well as uses of HMMs and RNNs in constrained, specific text generation, highlighting the intersection of machine learning and literary creativity while addressing the challenges of modeling complex language patterns.

2 Pre-processing (15 points)

The pre-processing stage was crucial in structuring the dataset for training a Hidden Markov Model (HMM) that generates Shakespearean-style poetry. Since sonnets follow a strict structure, we focused on cleaning and organizing the text while storing rhyming and syllabic information of the tokens for later use when generating sonnets. The dataset consisting of Shakespearean sonnets, *shakespeare.txt*, was carefully tokenized, formatted, and prepared to make the generated output both structurally and linguistically accurate. A wordcloud generated from all of the training data of the 154 Shakespearean sonnets is shown in Figure a, giving a visual representation of the most frequently occurring words.

We split up the text line by line, removing numerical sonnet labels and removing all punctuation except hyphens and apostrophes for a few words, as they were in the syllable dictionary provided to us. We chose to remove punctuation because it does not affect the rhyme or rhythm of the poem and could reduce the clarity of the generated sonnets if not learned correctly. Each unique word was represented by an integer, starting with (\n) at 0. Newline characters were added explicitly at the end of each line of words in the training sequences to mark line breaks, ensuring that the model recognized the structure of individual lines within sonnets. Apostrophes and hyphenated words were carefully handled so that words like 'tis and well-known were preserved with their linguistic significance and uniqueness. Each training sequence corresponded to a single sonnet, allowing the HMM to learn the connections between the lines within each sonnet during training. This step ensured that the training data maintained the poetic language used in Shakespearean texts.

This improved the quality of the generated sonnets by structuring the input in a way that the HMM could understand and replicate. By marking line breaks explicitly, the model was able to recognize where lines should end, allowing the HMM to learn the traditional 14-line structure on its own. Without these structured pre-processing steps, the model would have generated text without proper poetic form, resulting in an incoherent mix of words rather than a structured sonnet.

To ensure the generated sonnets followed Shakespearean conventions, we enforced rhyme patterns by extracting the last words of each line and mapping them in a rhyme dictionary. This dictionary stored all the last words in a line as a key and mapped it to all the words it possibly rhymed with in the training data. This allowed us to select from these rhyming words to seed the last lines of the poems later on. Additionally, we used a syllable dictionary created from parsing *syllable_dict.txt* to store the counts of syllables of each word that shows up in the 154 sonnets, including the different counts for when that word occurs at the end of a line.

Overall, we pre-processed the data in a way that made the sonnet structure clear and learnable by the HMM, but also stored data for enforcing some of the poetic constraints when generating the poem after training. Tokenization preserved line structure, the rhyme dictionary enforced correct rhyming patterns, and syllable constraints maintained proper meter. These steps created a high-quality dataset that enabled the HMM to generate a metrically accurate and well-rhymed Shakespearean sonnets.

3 Unsupervised Learning (20 points)

After preprocessing and tokenizing the dataset of Shakespeare sonnets, we continued with designing and training an HMM on these. We used the HMM class from the Homework 6 solutions because it allowed us to flexibly adapt the model to the specific requirements of our sonnet generation by modifying or adding functions to the HMM class. This included forward and backward algorithms for computing the alpha and beta probabilities, respectively, for each of the input sequences. On each training sequence, the likelihood of that sequence is calculated in the forward algorithm, then the backward algorithm calculates the posterior probability of the hidden states. These were implemented in the Baum-Welch algorithm for the unsupervised learning method in the HMM class. This algorithm maximizes the probability of an observation sequence by iteratively adjusting the transition and emission probabilities based on the computed forward and backward values. After each iteration, the model parameters are updated to improve the fit to the training data, increasing the overall likelihood of the observation sequences. We also had a supervised learning method that was used for placing constraints on the stress of syllables (see section 7). This labeled for the data were the stress on each syllable, so the HMM parameters are directly estimated from this.

We started with 4 hidden states and 100 iterations for training the HMM and got somewhat reasonable sonnet outputs from this. After trying larger numbers of hidden states (up to 15) we did not see an apparent improvement in the quality of the output (discussed more below), so we decided to stick with 11 hidden states as the optimal. Upon observing the top words for each state, models with more hidden states had very similar groups of top words for some states, indicating that these states may be redundant. Larger counts of hidden states took more time to train, so the tradeoff for compute was not worth the minuscule (if any) improvement in the coherence and sophistication of the sonnets themselves.

4 Poetry Generation, Part 1: Hidden Markov Models (20 points)

We used the unsupervised learning method from our HMM class as described above to generate a 14-line sonnet. Since we included ’0’ as the newline character, our algorithm for sonnet generation was to keep generating emissions until 14 ’0’ emissions were generated. This really tested whether the HMM learned the structure of the 10 syllables per line (because 10 syllables corresponds to 5-10 words). The lines had a variable number of lengths, as shown in the example sonnet below (from 6 hidden states and 200 iterations).

Something goddess but with may haply a tomb 11
For advantage have i this best with heart of a beauty being my head of his appetite 23
Whom you silvered love-suit to whose careful and fear 12
Not when thou thy firm shall april 8
Thy sire 2
And when on treads his face believed sit 9
Is his fresh with subsist words o first an thing 11
And ever with with way to eyes groans 9
For his lesser heaven’s fair winter’s skill only directly tattered 17
His most hate 3
You warrantise wretched not and looking lived to mine 13
Or his star can seen would rather with gentle form vilest to one my life speed bound of thee 22
Thee true better of your beard 7
For slander’s 3

The syllable count for each line was appended to the end of each of the lines (after it was generated), and we manually capitalized the first word of each line. Generating sonnets in this naive manner showed that the rhyme and syllable patterns were not learned by the HMM. The syllable counts are quite variable and none of the last words in each of these lines rhyme. In terms of retaining Shakespeare’s voice, the lack of coherence between the words themselves points towards a difficulty in learning English in the first place.

However, there is clearly some sense of the poem structure because some lines do make sense and none of them are excessively longer than 10 syllables. In many of the lines, there is at least one phrase that does make grammatical sense. An HMM is capable of learning these patterns due to its modeling of probabilities of transitioning between states. It likely learned that the probability of transitioning to a newline character is quite high for any given token, as there are newline characters quite frequently throughout the words in a sonnet. This would explain why the generated sonnets have reasonable line lengths, even if the syllable count and rhyme scheme are not accurately preserved. An HMM’s modeling of emission probabilities results in these syntactically plausible phrases, as it learned that certain ones are more plausible. Additionally, the HMM accurately learned about common words in English, as words like ’when’ and ’his’ are emitted quite often. Due to the use of unsupervised learning, it approximates the surface-level properties of a sonnet but has a harder time learning the nuances of the specific rhyme scheme or the exact number of syllables in a line.

Training with a larger number of hidden states actually resulted in sonnets that adhere to the syllable count more closely. These sonnets did come up with phrases that made more sense, but also seemed to use a smaller variety of words in order to accomplish this.

5 Poetry Generation, Part 2: Recurrent Neural Networks (20 points)

For our RNN, we implemented a character level LSTM using Pytorch. Our model consisted of an input layer that processed one hot encoded character sequences, all with a length of 40 characters, followed by a single layer LSTM with 128 hidden units, and a fully connected output layer that mapped the hidden states to probabilities for each of the unique characters. We also employed a softmax activation function in order to generate probability distributions over the different characters, using Cross Entropy Loss as the loss function and the Adam optimizer. We also did the necessary data pre processing by obtaining overlapping subsequences (again of length 40) using a sliding window approach, with a step size of 5 to speed up training (semi-redundant sequences). For simplicity, we fixed the number of layers and batch size to be 1, and chose 128 hidden units to be within the given range of 100 to 200. Initially, we set our learning rate to be 0.001, but with further trial and error testing across various ranges, we settled on a learning rate of 0.006 to enhance convergence without causing too much oscillation in the loss function. Finally, as suggested by the instructions, we introduced some variability in the text generation by applying a temperature parameter, scaling the logits before the softmax activation. In general, the temperature controlled the randomness of the output, where lower values showed more deterministic results and higher values increased the variability.

We tested our model at temperature values of 1.5, 0.75, and 0.25 using the seed text: ”shall i compare thee to a summer's day?”. When the temperature was 1.5, the generated text had a lot of randomness, frequently producing incoherent character sequences. At 0.75, there was improved coherence with some recognizable words emerging, but no clear sentence structure, Finally, at 0.25, the model generated repetitive, but more structured outputs. It favored frequently seen words such as ”thee,” ”the,” and ”me.” An interesting note was that at earlier trained epochs (around 20), we saw outputs with excessive spaces and random characters, but as training progressed, the model improved significantly in punctuation placement, particularly with spaces and commas. By the end of 100 epochs, while the generated text was still very repetitive, it demonstrated an improved ability to structure words and insert punctuation correctly.

Compared to the HMM, the LSTM struggled to generate complete sonnets or even recognizable sentences due to its character-level nature. While the HMM at least formed full length poems with recognizable words, its reliance on transition probabilities introduced many more grammatical mistakes. In contrast, the LSTM modeled better sequential dependencies, but required significantly more training data. While it captured the local character sequences effectively, it failed to learn the higher order structure necessary for generating meaningful poetic lines. So in general, the LSTM model exhibited some strengths in character prediction and punctuation placement but was ultimately unable to generate structured Shakespearean sonnets.

6 Additional Goals (10 points)

To improve the quality of our generated sonnets, we focused on refining one of the fundamental aspects of Shakespearean poetry: rhyme. While our initial HMM could generate sequences of words, the output lacked structured rhyme patterns, leading to poetry that did not fully resemble Shakespearean sonnets. To address this issue, we implemented a rhyme-enforced generation process that ensured proper rhyme pairings while maintaining coherence in the generated lines.

Since Shakespearean sonnets follow a fixed rhyme scheme (ABAB CDCD EFEF GG), we modified our poem generation process to seed the end of lines with emissions for rhyming words. To do this, we extracted the last words of each line from the training data and built a rhyme dictionary that mapped words appearing in corresponding rhyme positions. This dictionary stored rhyming relationships learned from the dataset, so that rhyming words were derived solely from the way Shakespeare used the rhyming in his sonnets.

Instead of generating lines in order, we selected the last word of each line first, ensuring that it matched its necessary rhyming counterpart before making the rest of the line. The remaining words were then generated in reverse order, until a ′0′ or newline emission was generated. This method prevented the model from producing mismatched or randomly placed rhymes and allowed for structured poetry generation. An example sonnet generated by this enforced rhyming process is shown below (from an HMM trained with 6 hidden states and 100 iterations) with syllable counts for each line at the end of each line:

Hopes in thine minds 4
Better the of self-killed in coral i no in ′tis if is you 16
Winds 1
Grew 1
World of day verse 4
Heart the look that wilt if green the worthless now when hence which can i you thy land my pure spring the cure time went with in that moan 30
Old an he eyes i inhearse 7
Gone 1
Ill that name that so lays breast 7
Once and in sweet the stol′n since oft view senses lips in of fame 15
Guest 1
Decay and name 4
When pleasant feature 4
Blood to thou why moods thy aught which there which why ranged is fairer thou joy restore whilst should till laid root lovers creature 28

This process worked in terms of generating 14 lines that are not excessively longer than 10 syllables that follow the abab cdcd efef gg rhyme scheme, but there is still a lack of coherence amongst the words. This incoherence may have been perpetuated by the backwards generation and forcefully inserting only rhyming words at the end of the lines.

One challenge we encountered was the backwards generation of emissions, since the model was trained on sequences in their forward direction, which meant that the transition probabilities were not naturally aligned for reverse generation. Since there were a limited number of words that rhyme, this heavily constricted the options for the ending emissions of each line.

Despite these challenges, incorporating explicit rhyme constraints significantly improved the structure of the generated sonnets. The rhyme dictionary allowed the model to produce consistent and predictable rhyme pairings, leading to a more Shakespearean feel.

7 Visualization & Interpretation (15 points)

. To better understand how the HMM captured patterns in Shakespearean poetry, we analyzed the learned states, transition probabilities, and word distributions. Our visualizations of the observation and transition matrices in Figures b and c show that both of these are quite sparse, indicating that the HMM learned a clear connection between states rather than jumping erratically between them (evidenced by the somewhat coherent and poem-structured sonnets that were generated). The sparsity of transitions indicated that the HMM did not transition freely between all states but instead followed structured movement patterns, reinforcing the poetic form. The sparsity of the observation matrix further confirmed that each state had specific word emission probabilities, demonstrating how the HMM captured relationships between words.

We produced wordclouds for each of the 11 hidden states in our final model, which show the more important words for that state in bigger font, as seen in Figures f - k. This allowed us to identify how various hidden states were associated with distinct clusters of words, revealing patterns in the structure of Shakespeare's text. These wordclouds for each hidden state showed that the model learned thematic groupings—some states contained words related to love and emotion (love, heart, self), while others were dominated by archaic verb forms (doth, hath, eye) or time-related words (day, thee, night). These groupings suggest that the HMM could learn and represent the different linguistic components of a sonnet. The top 10 words for some of the 11 total hidden states are shown below:

state 0: ['love', 'eyes', 'eye', 'heart', 'own', 'time', 'more', 'world', 'sight', 'day']
state 1: ['in', 'with', 'of', 'from', 'for', 'all', 'to', 'is', 'thou', 'on']
state 6: ['so', 'me', 'which', 'mine', 'thee', 'no', 'a', 'you', 'than', 'her']
state 9: ['thee', 'be', 'is', 'i', 'you', 'still', 'now', 'give', 'away', 'with']
state 10: ['of', 'self', 'to', '\n', 'sweet', 'thou', 'as', 'love', 'thoughts', 'all']

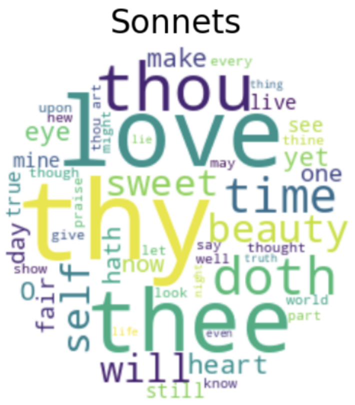
These reflect the fact that the HMM learned a general semantic structure of the words, because the top words in each state follow a common theme. State 0 most likely corresponds to love and romantic imagery, state 1 provides structure and connection in the generated text, state 6 could relate to personal reflection, state 9 pertains to action and resolution, and state 10 may be a mix of affection and resolution words.

To further analyze the HMM's learned states, we created a heatmap for the state transition probabilities that show the correlation between states for all possible combinations in this model (121 for 11 x 11), as seen in Figure e. The correlation between state 1 and state 5 is 0.97, which is reasonable because the top words for state 5 are: ['the', 'thy', 'my', 'a', 'of', 'thee', 'your', 'his', 'their', 'thou'] which are possessive words that would naturally come after connector words (state 1). The correlation between state 8 and state 3 is quite high, at 0.82. This makes sense considering the words most associated with each state. State 8's top words include ['\n', 'that', 'have', 'by', 'love', 'self', 'not', 'this', 'do', 'with'], which are commonly used in contexts like statements or reflections. On the other hand, state 3's top words are ['and', 'i', 'that', 'but', 'my', 'for', 'to', 'when', 'the', 'which'], which are typical in connecting thoughts, conditions, or contrasts. The overlap of words like 'that' and 'with' across both states, combined with the typical linguistic structure of English, warrants the high correlation between these two states.

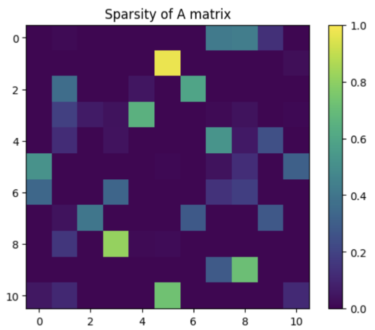
Additionally, analyzing the part of speech distributions across states revealed that some states were more likely to generate nouns and adjectives, while others contained verbs and function words, reflecting the syntactic organization of Shakespearean poetry. As seen in Figure d, the singular noun tag (NN) is most present in state 0, which we discussed earlier, corresponds to words about love and romance which usually refer to one person. The preposition or subordinating conjunction tag (IN) has a high frequency in state 1, aligning with the prepositional phrases that mostly made up the top 10 words in that state. Similarly, state 4 has close to half the distribution of the personal pronoun words (PRP), aligning with state 4's top words too: ['i', 'thou', 'you', 'it', 'love', 'do', 'he', 'they', 'am', 'will']. It is clear that the HMM learned, not only the semantic relationships between words, but also the syntactic patterns characteristic of Shakespearean sonnets, evidenced by the distinct part of speech tag distributions across states.

Through this visualization and analysis, we observed that the HMM effectively captured linguistic patterns in Shakespeare's works, learning meaningful state representations that contributed to the generation of structured, stylistically appropriate poetry.

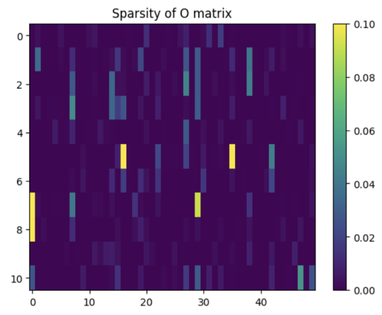
Relevant Images



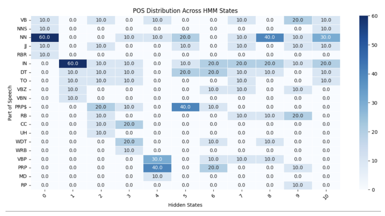
(a) WordCloud of Training Data



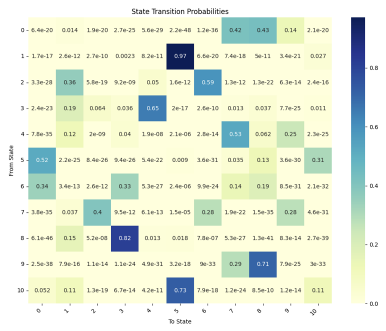
(b) Sparsity of State and Transition Matrices



(c) Sparsity of State and Transition Matrices

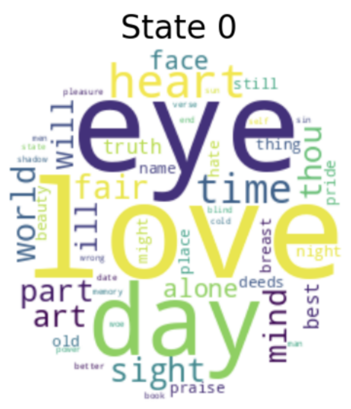


(d) Heatmap of POS Distribution

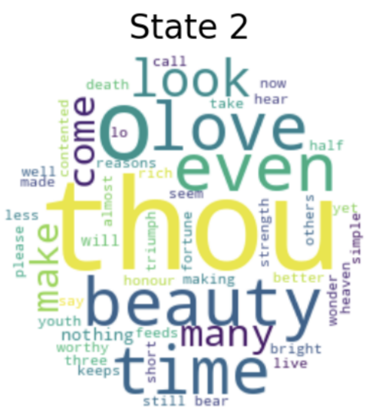


(e) Heatmap of State Transition Probabilities

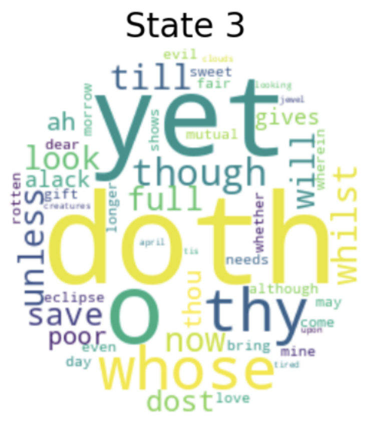
A Few Interesting WordClouds



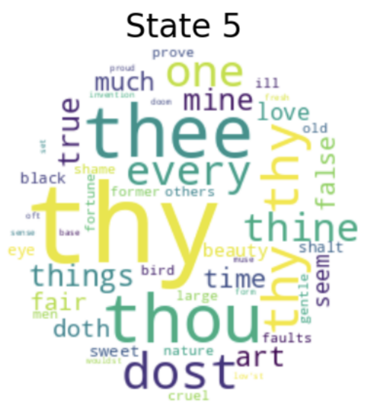
(f) State 0



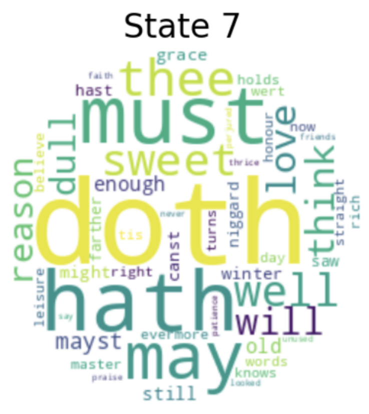
(g) State 2



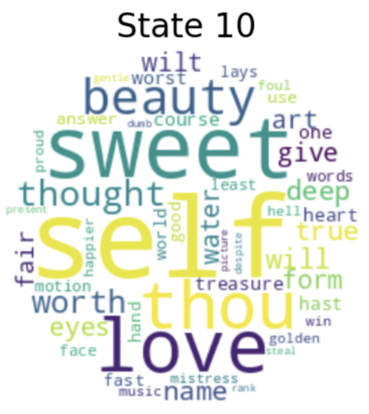
(h) State 3



(i) State 5



(j) State 7



(k) State 10

8 Extra Credit

For extra credit, we focused on improving the meter of our generated sonnets by enforcing iambic pentameter, the rhythmic structure commonly used in Shakespearean poetry. While our initial model ensured that each line contained exactly ten syllables, it did not account for the alternation between unstressed and stressed syllables that defines iambic pentameter. To address this, we refined the hidden states of our HMM to explicitly encode stress patterns and modified the transition matrix to enforce proper alternation.

To represent stress patterns, we categorized each HMM state as either stressed or unstressed, based on whether its index was odd or even. Using state_stress_label(), we assigned stress labels to states and then modified the transition matrix (A) in apply_meter_constraint(hmm). This prevented the model from transitioning directly between two stressed states, ensuring that each stressed syllable was followed by an unstressed one, thus enforcing the alternating rhythm required for iambic pentameter.

For poem generation, we used sonnet_generate_emission_meter() to ensure that each generated line followed the expected syllable count and stress alternation. This function iteratively sampled words while checking their syllable count and stress type, discarding words that would break the pattern. If a selected word did not fit within the constraints, it was removed, and a new word was chosen. This ensured that the final output adhered to both the ten-syllable structure and the correct stress alternation, making the generated text more rhythmically accurate.

While implementing strict metrical constraints reduced the variety of possible word sequences, it significantly improved the authenticity of the generated poetry. By enforcing stressed and unstressed syllable alternation, we brought our generated sonnets closer to the traditional flow and cadence of Shakespearean verse, making this a valuable enhancement to our HMM-based poetry generation.