Machine Learning & Data Mining
Miniproject 2: MovieLens Visualization

Caltech CS/CNS/EE 155
Released February 16st, 2024

---

1 Introduction

Team name: Beavers
Names: Brendan Flaherty, Sujit Iyer, Sayuj Choudhari
Work Division: Worked together on all parts of the project.
Packages Used: pandas, numpy, matplotlib, seaborn

2 Pre-processing

https://colab.research.google.com/drive/1B_yEhytMV0q-XVrpPuW7hhfcgQpYSYGw?usp=sharing

We developed two different pre-processing methods, one for the Hidden Markov Model and one for the Recurrent Neural Network. In general, for both methods we saw removing punctuation to simplify model training to make a significant difference on the model's performance in terms of how much sense generated lines made. To start pre-processing for either method we first generated 3 datasets that would make later pre-processing steps easier: parsing the observations into a indexed observation list, an array of words, and an array of each word and relevant information to that word. The first two datasets were generated using the observations parsing function from Homework 6, while the second was made using a combination of those results and the syllable dict. The poem observation array helped associate words to indices which would be helpful as numerical data for training the transition matrices in the HMM. Additionally, the poem map would help map the words to their respective indices so emissions from the HMM can be converted back to words. The third array developed was a more detailed version of the poem map with additional entries for each word in order including its possible syllable values (from the provided syllable dictionary) and an indicator variable on whether that word in order was at the end of a line (helps with rhyming).

HMM Pre-processing

Two main steps were helpful for pre-processing data to result in better training of the HMM. First a rhyming dictionary was developed to link rhyming words so that a rhyme scheme can be generated. This was done by accessing data rows in the third data array where the word was indicated to be the last word in the line. From there, we could use the rhyme scheme of a sonnet to iterate through the words and build a dictionary that keys each word to all words that rhyme with it. Notice for rhyme scheme generation that we need to build two independent rhyming sentences and then write those in the sonnet with some given order. This is done by reversing the generation by determining the rhyming pair of words first, then predicting the state that word was emitted from (done by maxing the state that would lead to that observation in matrix O) and generating forward from there. However notice if we want to build the sentence backwards and then reverse it, if A and O are trained on forward grammar data (normal sonnets) then it will generate forward grammar, and reverse it into grammar that is likely to make no sense. As a result, we felt it best to instead take the parsed poem observations and reverse each line, so that the model trains on a backwards grammar so when the generation is reversed for the rhyme scheme (basically double reversing the generation), the grammar is now forwards generating and will make sense. During testing, this choice seemed to work as reverse observations generated poems that were more coherent.

RNN Pre-processing

Relative to the HMM Pre-processing the RNN pre-processing was focused on character generation rather than word generation. While no punctuation was added as explained earlier, the RNN did train on spaces and new line characters as it was not directly generating words. More so, instead of a word dictionary used for emissions, a character dictionary was developed from iterating through all sonnets to key all characters to some index that would be used for training. For training, the RNN was trained on data input being all characters iterated through in all provided sonnet data, and the output being all characters successive to each character in the training input data. One step we took in manipulating the training data was tokenizing it as a all-characters-length one hot vector to represent each character. The reasoning for this is that neural networks mathematically calculate multiple dot products on input data, so for qualitative data such as characters, each character should carry the same weighting. For example, taking $A = 1, B = 2, C = 3$ as input into the RNN implies $2A = B, 3A = C$, which is not true. Instead $A = \{1, 0, 0, ...\}, B = \{0, 1, 0, ...\}, C = \{0, 0, 1, ...\}$, keeps the weighting (defined by magnitude) of each character the same and each character's representation orthogonal to all others, which implies that all characters are equally different which is also true in a qualitative sense. These two steps of generating input data as character lists of observations and then tokenizing each character as a one-hot vector aided in model training.

3 Unsupervised learning

https://colab.research.google.com/drive/1B_yEhytMV0q-XVrpPuW7hhfcgQpYSYGw?usp=sharing

We did not use any special packages to create our HMM for poem generation. We instead used the Baum-Welch algorithm that we previously implemented in HW6. Based on our analysis in HW6, we deduced that increasing the number of hidden states would also increase the coherence of our poem generations. Similar to HW6, we tested with 1, 2, 4, and 15 hidden states and saw that our poem phrases became increasingly more realistic. Therefore, we decided to use 15 hidden states for testing and submission purposes. To ensure that our poems and their coherence were reproducible, we created the unsupervised HMM several times and saw similar results. Due to time constraints, we did not use more than 15 hidden states.

4 Poetry Generation - Hidden Markov Models

https://colab.research.google.com/drive/1B_yEhytMV0q-XVrpPuW7hhfcgQpYSYGw?usp=sharing

Our algorithm for generating the 14-line sonnet was contained in the function naive_sonnet_hmm. We had 3 parameters for the function: 1. hmm (The HMM trained using the unsupervised_learning function developed in HW6), 2. obs_map (the poem map from words to a unique index created during the parsing of the text), and 3. syllable_dict (a map from a word to the number of syllables it contains). The body code loops 14 times, calling sample_sentence (from HW6) each time to generate the 14-line sonnet. Specifically, the algorithm first ascertains whether or not it is the start of a new quatrain/couplet. If so, it adds a new line to the sonnet to ensure proper spacing. A sample sentence is generated using a randomly chosen number of words from 5 to 8 (We knew this range of words was likely to generate lines with close to 10 syllables). The

sample sentences are generated based on our probabilities of transitioning between hidden states (representing underlying poetic structures) and emitting observable states (specific words or syllables). After the sample line is generated, we count the number of syllables by consulting syllable_dict in a helper function count_syllables. If the number of syllables is not 10 then we repeatedly generate new sample sentences until it is. Eventually, a 14-line sonnet is generated. Below is an example of a sonnet generated with naive_sonnet_hmm using 15 hidden states.

```
Leisure on your plea for reproving the
To thou your forsworn shall poverty should
Crime i my greatest elements eyes proved
The better ocean sweet a can honour

Ruining do vices leap on find with
So well than swallowed misplaced of selfsame
Sweet rider darling is scorned lease wretched
Love and of quickly sorrow thy gentle

Thee ill within lame painter report the
Saucy feeble love is despise none one
Of unless a registers so thee a
Rehearse sword stands the extern tyrants breath

Moment son for thee me husbandry in
Unless muse and the shore yet immortal
```

To generate a sonnet, we utilize the naive_sonnet_hmm function, which aims to create each line of the sonnet by sampling words based on the trained HMM, ensuring each line adheres to a predefined syllable count, typically ten syllables, resembling the structure of a Shakespearean sonnet. The generation process doesn't explicitly control for rhyme or rhythm, focusing instead on the syllable count and the probabilistic selection of words.

Evaluation of Generated Poems:

We observe that the quality of generating poems in a naive manner is generally pretty poor. Because the generated sonnets do not explicitly control for rhyming, it leads to poems that lack the consistent ABAB CDCD EFEF GG rhyme scheme characteristic of Shakespearean sonnets. Additionally, while the algorithm maintains a standard ten-syllable count per line, the rhythm often deviates from the iambic pentameter typical of Shakespeare's work. This discrepancy arises because the model selects words based on their occurrence probabilities and syllable counts without considering stress patterns.

A reason why the poems do not make much sense is also that each line is unrelated to the next. Each line is generated by calling sample_sentence, which generates an emission assuming that the starting state is chosen uniformly at random. In the future, we could preserve the current hidden state to create text longer than just one sentence. The poems seem to retain Shakespeare's original voice given their antiquated and flowery word choice, however, it is still difficult to capture Shakespeare's essence without the capabilities to reflect his depth and meaning.

We see that a lower number of hidden states tends to produce more repetitive and less coherent lines. A higher number allows for greater diversity and complexity in word choice. This suggests a correlation

between the number of hidden states and the model's ability to capture and reproduce the underlying patterns in Shakespeare's works. Logically, this makes sense given our knowledge of how HMM works. With 1 hidden state, the word choices are randomly generated based on the word frequencies of the sample text, and we can't pick up on semantic patterns. We can form more coherent phrases as we increase the number of hidden states.

5 Poetry Generation - Recurrent Neural Networks

https://colab.research.google.com/drive/1B_yEhytMV0q-XVrpPuW7hhfcgQpYSYGw?usp=sharing

To create the character-based LSTM model, we used the pytorch library. We initialized a model with a single layer of 100 LSTM units (nn.LTSM), followed by an output layer (nn.Linear) with a softmax nonlinearity (nn.LogSoftmax). While creating our 'CharLSTM' module, we followed the provided RNN tutorial which provided ample support in creating the init, forward, and initHidden functions. To train the model, we minimized categorical cross-entropy loss and used the Adam optimizer. We trained the model for 100 epochs in batches of 500 sequences, using reusable code from HW4 (enumerating the training loader and accumulating loss). Every 5 epochs, we took the current model, and evaluated its performance on the 40-character seed "shall i compare thee to a summer's day?". To do this, we created a 'predict' function that took the natural exponent of the log softmax model outputs for the 40-character seed (During training, we realized that taking the log softmax as our nonlinearity improved the cohesion of the poem generation.) We generated 490-character-long poems because we assumed that the average line with 10 syllables contains 7 words and each word has 5 characters (5 * 7 * 14 = 490).

During training, we tuned the learning rate parameter. We noticed that a learning rate of 0.01 was optimal for time efficiency and cohesion of the generated poems.

Below are the poems generated after the final epoch using various temperature parameters:

```
Temperature = 1.5:
shall i compare thee to a summers day
that doil to make on you pery of bered
this aphomname
of thy chariops of prepart of side
which bosulpoar dury as minded on my deed
by then find out many matiebsby joxed bester rieming love knows my public kirs our matter
be lines times fill as minders better loving
pkiden garters supprabige of mone leases
he hapl aught if his which fair name out i have needy
the vooked once upon my badsmed atherbd
yet singled carer is turn cur makin thee quent sing


Temperature = 0.75:
shall i compare thee to a summers day
but saves my verse is as wouldsh in your compare
my self it must and in age nor is slandered at the prepailed the clead so rugnss and all my self are so
hearter where bur unmecture the relvest on thus fillound are not i seee
that thou dost live and truetious prime
therefore than will are resting a maker holds her heart
where do not fauths seef that my heart the age
and summer in me
and vowartants and do thy sweet favour book if i thine
shall seem t


Temperature = 0.25:
shall i compare thee to a summers day
that i am not the profane so rehearse
that have i do contieri me a chester and in his love her but in love with self dost thou deserved accope
therefor in thy summers dear
and even so shauly forgot to me under nee
in the day them the lease the ranks and in my verse earth do i not poor silled
when i see their state
and subject thee to be as a sooker character
which like a spiot on dost flowers doth come too
thy self alone with thy sweet forgetel
to
```

We can see that the temperature parameter has a significant impact on poem generation. Higher temperatures generate more complex and varied words but mostly generate words that make no sense. This is intuitive because the longer the model tries to chain characters to form a word, the more likely it is to string together characters that result in nonsense. The small temperatures generate simple words which are often repeated but the poems resemble a Shakespearean sonnet the best.

Overall, we can see that the poems generated by the LSTM model are poor, and often contain gibberish. This is because our model is character-based. Having a character-based model makes it difficult to learn sentence structures because sentence meaning is derived from word combinations rather than character combinations. A character-based model is too abstract to generate realistic poems. The quality of the poems for the LSTM model is much worse than the quality of the HMM poems. The LSTM needed much more training data and the runtime was 10 times longer than when generating poems with the HMM.

6 Additional Goals

https://colab.research.google.com/drive/1B_yEhytMV0q-XVrpPuW7hhfcgQpYSYGw?usp=sharing

Alongside the poems generated by the naive HMM, we implemented an HMM with a classic sonnet rhyme scheme. After building the initial HMM, it was quite simple to implement a rhyme scheme. We used the rhyming dictionary mentioned earlier in the HMM Pre-Processing section to link rhyming words. To generate an entire rhyming sonnet, we split the function 'generate_rhyming_sonnet' up into 3 calls to the function 'generate_quatrain' and 1 call to 'sample_rhyming_couplet'. 'sample_rhyming_couplet' randomly selects a word from the rhyming dictionary and then randomly selects a word that it rhymes with as its pair. Independent sentences with the rhyming words at the end are generated through a call to the function 'generate_rhyming_emission'. This function takes the observable state of a rhyming word and predicts the current hidden state by maxing the state that would lead to that observation in matrix O. A sentence is then created similarly to the regular generate_emission function (Using our O and A matrices to randomly choose a new state and a new word emission). We added syllable count control to the function so that emissions are continually generated until we reach 10 syllables for the line. At the end, we reverse the emissions because our rhyming word should be at the end.

As previously discussed in the preprocessing section, if we wanted to generate the sentences in the reverse direction, it made more sense to reverse each line of the parsed poem observations, so that the model trains on a backwards grammar. This is because 'generate_rhyming_emission' randomly selects the next state and the next emission for that state but we want the 'next' state to be the 'previous' state. This makes it important to properly train our A and O matrices affiliated with the HMM object. The following poems were generated by the function 'generate_rhyming_sonnet' for hidden state parameters of 1 and 15.

1 Hidden State

```
I to and a thee what though betwixt any
Thoughts those hath been brought the all fawn brow moan
That a due the becomes he then not many
Chase we in his is eternal stay gone.

Mine against am all though gilded sight fiend
Yet not that a therefore then the think tired
My that only separable mud threw
O or their epitaph fester expired.

Before flesh live cure fear be night arrest
Thou in as they is those ruminate strive
Me haply be such firm thee hide interest
Time must sorrow world that thou style dost alive.

Put mine once friend swallowed disgrace devised
Foul which have which turned exceed sympathized.
```

15 Hidden States

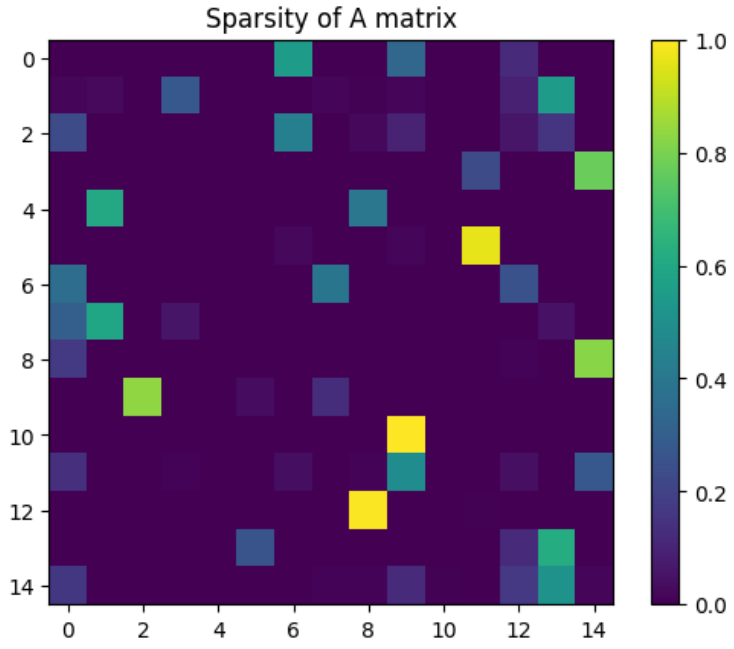
```
I how yet me to in any madness sake
And thou excuse nor them I can strive fiend
Partly sorrow away any sight with in come
Sullen eyes hope one fall rudely carved threw.

Onset not worth I thee I for woe loving
Transfix my love of holds world bred favour story
Did glorious lose never nourished who reproving
The which it impair together other glory.

This enough nor to to my beauties tongue
In quest what behold me hush fast ruth you
The nought did heat thee for love stained unjust
Seconds the death pluck after but within.

Well to from thy rhyme is uprear untrue
Tie not tongue catch due spites the smell flattery.
```

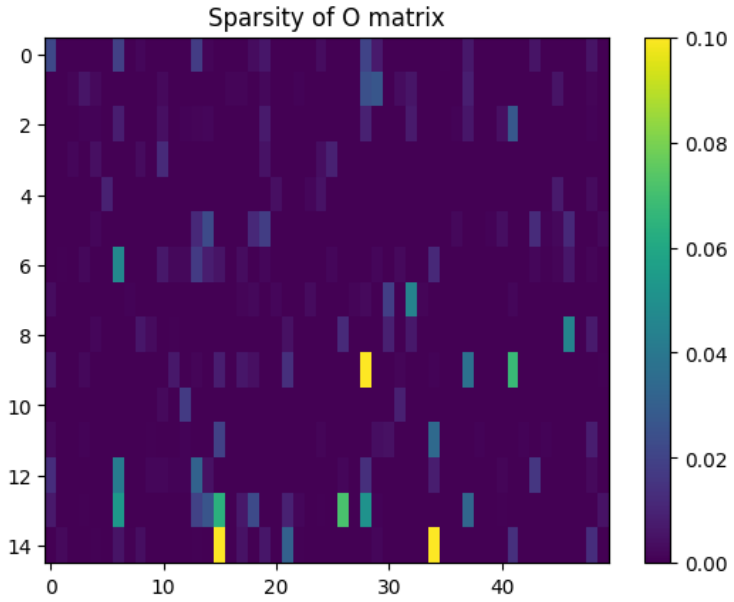
Here we see that the poems still do not make much sense. This was expected because the goal was to emulate Shakespeare's voice. We did not directly change how sample lines were created other than generating emissions backward.

7 Visualization and Interpretation

https://colab.research.google.com/drive/1B_yEhytMV0q-XVrpPuW7hhfcgQpYSYGw?usp=sharing

As in HW6, we created visualizations for the sparsity of the A and O matrices. Our sparsity visualizations were done on our HMM with 15 hidden states and reversed training data.





As we can see, both the transition and the observation matrices are quite sparse. This means that there are very few realistic transitions the model can take at each timestep. Furthermore, there are few possible emissions for a given state.

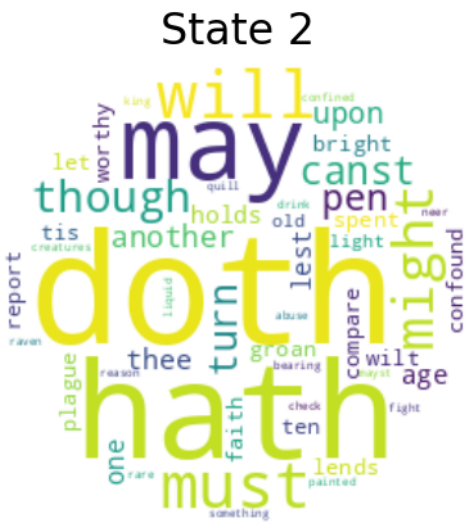
Take, for example, the possible transitions from state 10. We see that state 10 can only transition to state 9 (the colormap tells us the probability of any other transition is roughly 0). If a transition to state 9 does

occur, the visualization of the sparsity of the observation matrix tells us that emissions 27 and 41 are most likely. Further analysis of the qualitative descriptions of each state could reveal why this happens. Overall, this is convenient for test purposes because the transition and observation behavior at each state will largely stay the same from iteration to iteration, but it also could lead to poor generalization.

To visualize the top words for each hidden state, we used the word cloud functions from HW6 but added a slight modification. We edited the states_to_wordclouds function to return a dictionary mapping each state to the list of its top 10 words and analyzed. Below are the word clouds for 5 of the 10 states as well as their 10 most popular words.

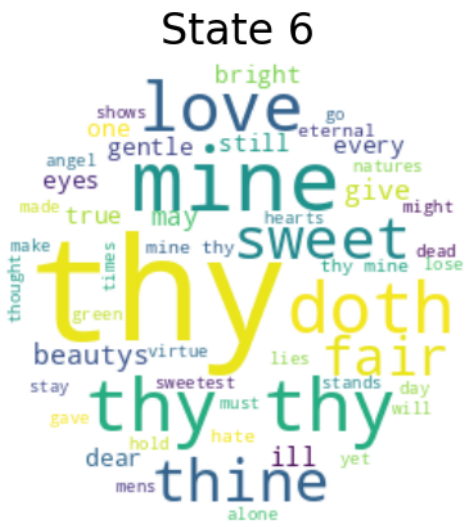
Top 10 words: ['love', 'eye', 'part', 'beauty', 'praise', 'best', 'heart', 'day', 'thee', 'one']

Analysis: All of the words in this state focus on love and beauty, with words directly associated with romantic admiration



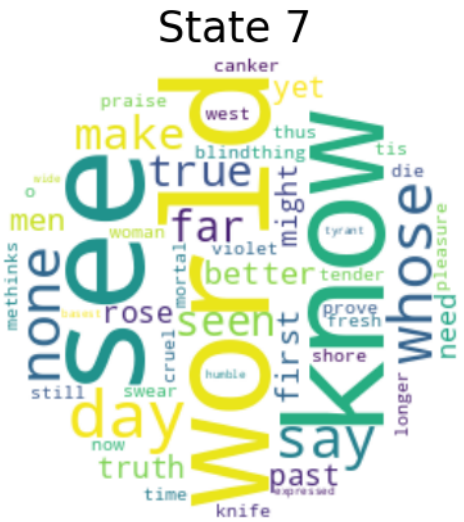
Top 10 words: ['doth', 'hath', 'may', 'will', 'though', 'must', 'might', 'another', 'turn', 'upon']

Analysis: The words in this state are dominated by modal verbs ("may," "will," "must," "might"). This state reflects themes of potential actions, decisions, and uncertainty or speculation.



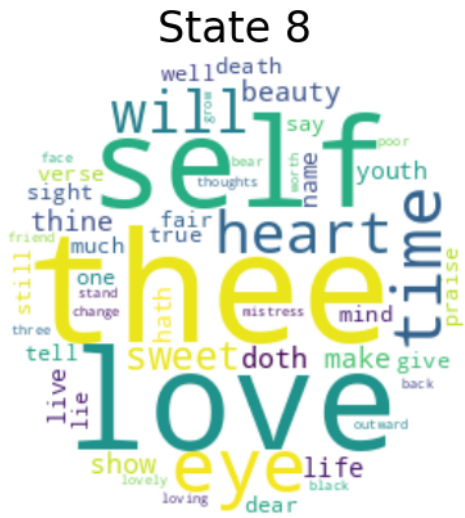
Top 10 words: ['thy', 'mine', 'doth', 'love', 'thy thy', 'sweet', 'thine', 'fair', 'beautys', 'ill']

Analysis: This state contains several possessive pronouns ("thy", "mine", "thine") as well as words of affection like "love" and "sweet", reflecting a theme of personal relationships.



Top 10 words: ['say', 'world', 'know', 'far', 'whose', 'see', 'day', 'seen', 'make', 'men']

Analysis: This state contains expression and perception verbs (”say,” ”know,” ”see”). It reflects how we perceive the world and communicate our observations.



Top 10 words: ['thee', 'love', 'self', 'eye', 'heart', 'time', 'beauty', 'sweet', 'doth', 'will']

Analysis: These words are very similar to state 1 in that they focus on love and beauty. State 8 also adds the word ”self” which merges themes of love and self reflection.

Finally, we used the animate function from HW6 to visualize the process of transitions and observations between states. The darker arrows indicate a higher probability of transition between states. Below is the

final frame of the animation, with the final emission being "As thy hell and foison sweet so are."