Machine Learning & Data Mining
Project 3 (HMMs for Sonnets)

Caltech CS/CNS/EE 155
March 15th, 2025

Introduction

- Group members: Amitesh Pandey and Jafarbek Arifdjanov
- Team name: Amitesh Pandey and Jafarbek Arifdjanov
- Division of Work: Mostly equal. Jafar worked on the Preprocessing and RNNs parts, and Amitesh worked on the visualisation and unsupervised learning parts. The report was a joint activity.
- Packages Used: Matplotlib, Numpy, NLTK, PyTorch, Wordcloud
- Colab Link: Code here
- Piazza Link: Post Here

LLM Usage

LLMs were minimally used. At no point in the project was a specific prompt passed to any of the major LLMs. The primary browser we used (Brave Studio) has an LLM engine incorproated into the search engine, which at times, directly displays code solutions that help debug. These were used from time to time but due to the nature of these results, it is not possible to retrieve each and every time the built in LLM was meaningfully assistive. Sorry!

1 Pre-processing

We preprocessed our data mainly using the NLTK library in Python. We divided pre-processing into two classes broadly, one to deal with syllable counts and the other specifically for sonnets. Here is a rundown on some of the main methods used in each of these classes

- **SyllableCounter**: For CMU's Pronouncing Dictionary
  - We parse each line to extract words and their stress patterns
  - We count each vowel as a syllable unless the word ends with an 'e.'
  - In case the word has no such letters, we simply use a syllable count of 1.
  - To get the stress pattern of a word, we take every othe syllable into account.
  - We remove punctuations at all instances
- **SonnetProcessor**: For Shakespeare's Sonnets
  - We tokenized the sonnets into lines and words with options such as `remove_punctuation, by_line, lowercase`
  - We analyzed syllable metrics per line such as average syllables per line, most common syllable, and the distribution of all syllables in Shakespeare's sonnets.
  - We analyzed stress patterns in the sonnets for matching the iambic pentameter.
  - We analyzed rhyme schemes in the sonnets by comparing them to expected schemes.
  - We created n-gram sequences for the training of the language model

2 Unsupervised Learning

We used the Baum-Welch algorithm as in HW6. This algorithm relies on an EM structure where the E step consists of backward and forward passes with α, β probabilities and expected counts whereas the M step is updating the observation matrices with E results.

In accordance with our findings in HW6, we found the number of states used in the production of our sonnets using this method to be directly proportional to the coherence, grammatical and logical accuracy of the generated sonnets. We tested `[5, 10, 15, 20, 25]` states and had `n_iters=100`, so our code was quite computationally expensive. Despite being at risk of overfitting, we found 25 to be the best one.

Packages Used

- Only `numpy` for random sampling in `generate_emission()`
- Core algorithms implemented in pure Python

3 Poetry Generation, Part 1: Hidden Markov Models

Algorithm

For our implementation of the naive HMM to generate 14-line sonnets, we relied on the Baum-Welch algorithm from HW6 has described in the previous section. Here are the specifications

- `generate_poem(hmm, num_lines)`: Takes an `hmm` object and generates a poem of `num_lines`.
- `unsupervised_HMM(n_states)`: Generates an `hmm` object with `n_states`.
- `class HiddenMarkovModel`: Holds the general implementation of the Baum-Welch algorithm.

Here is an example of a 14-line sonnet generated by our HMM:

```
Lame thou answered to now by my thoughts
Men straight foist be their way as it
For grow these him make proceeds nature then
Twain say so with wondrous out the verse

Prevailed critic ten time nor sweet not privilage
Sweet grace what goes leave this imprint of
Ill it owe building well dead something my
Forget thou wilt for thus then solemn some

Sin face sun no some orient and to
To some poison majesty good lead tears thee
Poor you i new speak that i saw
Out fell shall year so it art returned

Birds unseeing their antique twire of deny place
To thee to my years which thou make
```

This poem was realised through an `n_state = 25`. It is very close in its verbosity to Shakespeare's actual sonnets. We can clearly see an abundant use to Old English words like "twain,""thou," and "twire." The syllable count matches the expected syllable count which was set during preprocessing. The sonnet unfortunately **does not** follow a specific rhyme scheme. The sonnet **does not** make logical sense but it seems like it's talking about much of the same things that conerned Shakespeare, namely men and women, nature, sin, majesty, and art. This sonnet does retain Shakespeare's voice since the voice itself is largely a combination of the synactic preference, word usage and topic discussed. Since our generated sonnet also has the **syntactic inversion** that's commonly observed in Shakespeare's sonnets, such as beginning a sentence with "Lame thou," it closely models the voice of Shakespeare. As expected, increasing the number of states increases the coherence and grammar of the generated sonnet. I think the word-usage is obvious but the overall diction was easy to capture for the HMM through the stress pattern and syllable measures used.

4 Poetry Generation, Part 2: Recurrent Neural Networks

We used PyTorch for our implementation of our character-based LSTM model. We used a single layer and we had our embedded (initial) dimension set to 100 and the hidden dimension set to 150. We picked our `vocab_size` by going through the sonnets. We used a dropout within our initial layer. See:

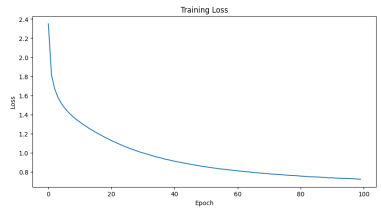


Figure 1: Loss vs num_epochs

Through Figure 1, we realised that the loss was broadly asymptotic beyond `num_epochs = 100` and settled on this value. We also tuned with varying initial and hidden dimensions and dropout probabilities. We generated poems using different temperatures as required. See:

Temperature = 0.25

```
shall i compare thee to a summer’s day?
thou art my all glory did praised the stars in seeming,
on hate a happy tongues is found a given to this store,
so that my name receive them to give?
profitles an painted best,
for i have sworn thee fair, heaven’s graces and that my tongue,
thy sweet beloved name no more shall summer’s purpose,
the in thy contined age,
when shall bear thy love’s pressance of this purious look,
let my love sweet self dost shape i have still,
that in the chronicle of thine eyes best jawny,
as thou art so deeds,
sings his guest,
my heart in thy self thy grace,
```

Temperature = 0.75

```
shall i compare thee to a summer’s day?
thou art they for her sorrow,
he in the stars hold i was day sin,
lily’s world would do so buried abused,
and the chery.
but when if it do my amed
to face worlds of that i be carvest,
and cheek of ourstancy bound:
for she hath her praising pen
will be a date,
but thou toivity beauty should be ad,
beauteous night by day of thy did love,
but you shall summer’s brave leaves in warrie,
the rained regished never!
```

Temperature = 1.00

```
shall i compare thee to a summer’s day?
thus shalt they see.
for i hopry this scarlet
from heat known their spirit in thy grace,
haw, and his love asholdough of thy amest.
thy glass and there are dead,
making a man i fair wholds my love, yea
take thee after not love gave thee vough as well seem,
but to die set forswert in onvern sad soil.
all time’s injurious days,
yet wherefore art
so bless you should do the sweet, but not with
a bastard shows
whereon your sum.
```

Temperature = 1.50

```
shall i compare thee to a summer’s day?
that i me! is never majesty,
and ten times ere
hath my slave who of feathered much more righth.
may see they incharves nor reking:
the body pateness
dreathed, but in the eye,
do i wofter of drinks
of your cold desire:
if too much ruinary? the
world true nothing on hidly face since sweet
seem hang praise taken that mine weed,
when i will or nature  you hate’.
with prouder to thy
```

Notice that the temperature parameter did not have a significant impact on the quality of our generated poems. This could be a signal that our RNN is prone to overfitting the Shakespeare sonnets provided, this is expected as our overfitting resistance is not that strong. As for the poems generated,

- **Vocabulary:** The poems are what you'd expect from Shakespeare. Old english cruft is maintained.
- **Synatax:** The inversions and run-ons typical of Shakespeare are still present.
- **Punctuation:** An excessive use of mid-sentence question marks and apostrophes is still present.
- **Rhyme:** The poems are **not rhyming**.

In general, the RNN performed considerably worse than the HMM. This can be explained by the fact that we are using character-level abstractions to generate sonnets and there is simply too much qualify information/signal loss taking place in doing so. The HMM on the other hand is able to benefit from clusters of words in its sonnet generation method.

As mentioned previously, both our models were rather computationally expensive. We realised that Google colab might be unreliable in running them and opted to run them natively instead. All in all, the HMM took ≈ 13 minutes to run and the RNN took ≈ 40 minutes to run.

5 Additional Goals

Code for Incorporating Rhyme

We opted for improving the rhyme of our generated sonnets as an additional goal. Here is a systematic breakdown of the functions we used and our technique to achieve this goal.

- `build_rhyme_dict`: Groups words into sets keyed by an approximate 'rhyme key', derived from the final stressed symbol in their stress pattern.
- `get_rhyme_key`: Returns the 'rhyme key' (final stressed symbol) for a given word, as a quick approximation for rhyming.
- `generate_line_with_meter`: Produces a single line by sampling from the HMM token-by-token, aiming for a total of 'target_syllables' syllables. Terminates when it exceeds or meets the target.
- `generate_rhyming_line`: Creates a line of about 'n_syllables' syllables. Tries multiple times if a rhyme_key is specified, ending the line on a word matching that rhyme key.
- `load_and_combine_texts`: Reads and tokenizes sonnets from Shakespeare and Spenser, merging their tokens into a single list. Returns the mapping from words to indices and vice versa.
- `generate_sonnet_rhyme_meter`: Produces a 14-line poem following a specified rhyme scheme (default is Shakespeare's), with lines aimed at n_syllables and forced to rhyme according to the scheme.

Analyzing the Generated Sonnet

Setting the `n_states = 10`, after preparing our new rhyme ductionary we called our `unsupervised_hmm` method described in the previous sections. The result was not disappointing. See:

```
Did forgoing to it void up god ever of portrait
Why cage is the may being soul love annoy
Beauty net of youth general not but elsewhere tie scorning
So fault easy so which of narcissus better

Proud in home to time best sins more letter
Something slide 's thou this fiend of it time time
And whom long love to annexed lines and made solitary
I self which be in will could die why which

That to depend her the may part sweet down myself
Although i tongue kindle breasts thought posterity thy beauty
Of love will my for thee power or to humbless
I the with those smart drop what mixed mantle truly

Thee art summer lovely oppressed desire separation never tract of
That yet love bud bait end doth make that to
```

We can see that the generated sonnet actually does have a fair bit of rhyme to it. Such as subsequent use of "letter" after "better," "truly" after "beauty," "of" and "to." Unfortunately, the rhyming scheme is not as immediate and requires a bit of recitation and accent modulation to catch onto and it is also compromising the coherence of our sonnet as expected.

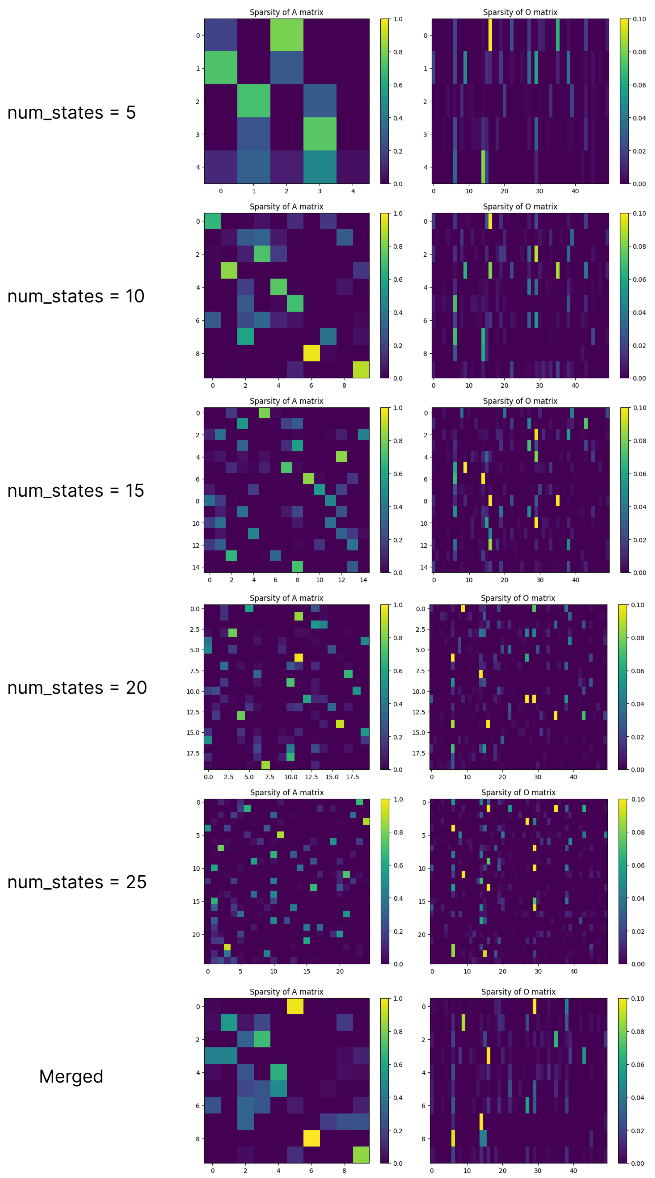
This somewhat poor performance could be due to the fact that there weren't enough rhyming combinations. In a case where a line ended with a word with no corresponding rhyming combination we generated a word randomly. Despite this somewhat trivial "hack," we could still generate sonnets with at least half a dozen rhyming pairs.

Theoretically, if you run the corresponding cell in the Colab enough number of times, you could generate a sonnet that rhymes perfectly. It's just a matter of probability.

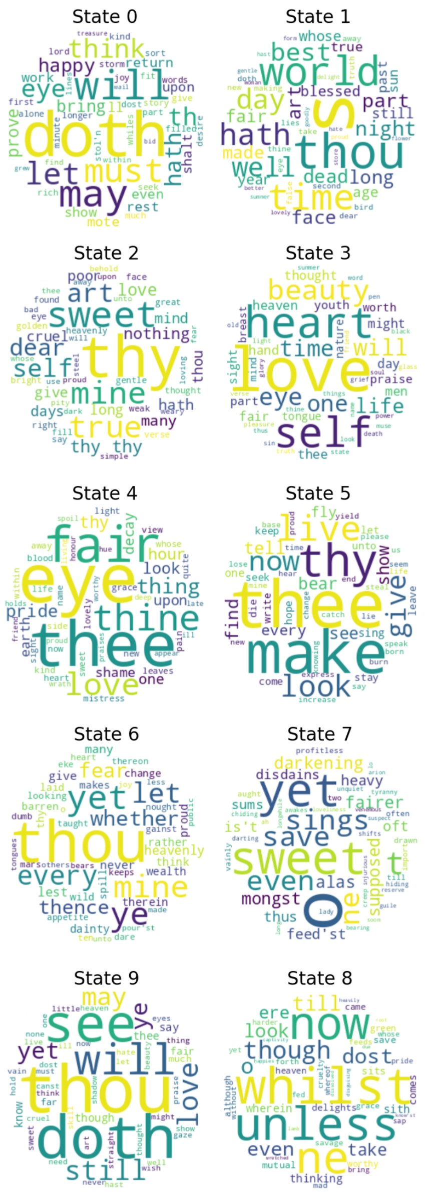
6 Visualization & Interpretation

Visualizations

We will now reveal the sparsity matrices of A and O as in HW6.

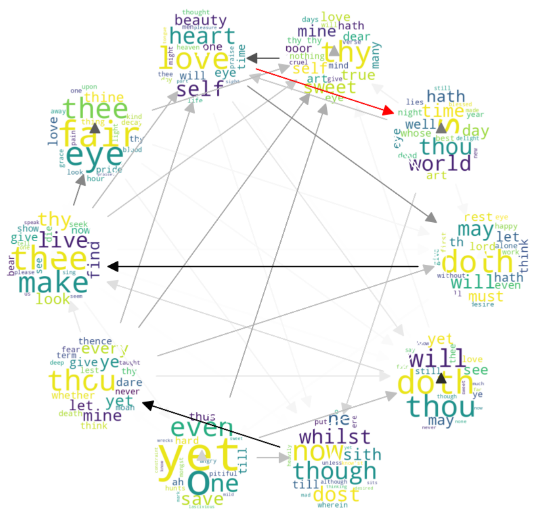


We will now reveal the wordclouds with different states as in HW6.



Now pay notice to the transitions between states in the Hidden Markov Model.

To be fading this unknown and a changing



Discussion

First notice that the A and O matrices are extremely sparse. This implies that there aren't as many transition signals in the data to begin with. As expected and discussed in HW6, an increase in the number of states induces an increase in the sparsity of both A and O. On average, A seems to be a bit less sparse than O, this is specially true at lower numbers of states. Analyzing the matrices at a given column makes us conclude that there are a limited possible emissions at any given state.

As for the wordclouds of different states, notice that states 0-2 have an overrepresentation of 1-syllable words. States 3-5 have an overrepresentation of words that align with beauty, attraction and love, which is a common theme in Shakespeare's sonnets. State 7 has words such as "disdain," "darkening," which could be the opposite of the previous states. Finally, by state 9, we have reverted to the 1-syllable pattern observed in the earlier hidden states.

From the state transitions diagram, we observe a specially strong connection between the 1-syllable states, indicated by a darker colored arrow. We can also see that the themes of "love, beauty, and heart" are strongly connected (and contrasted with) the themes of "lies, dead, and time." The state in which "make" and "thee" are popular is fairly strongly connected to the state in which beauty-objects like "eye" or "love" are popular. This could have to do with there being an excessive description of love and its objects in the sonnets of Shakespeare. All in all, these results have met expectations of Shakespeare's sonnets.

Most Used Words

Table 1: Most used word probabilities for States 0-9

State 0		State 1		State 2		State 3		State 4	
Word	Prob	Word	Prob	Word	Prob	Word	Prob	Word	Prob
to	0.1848	’s	0.0888	my	0.1098	the	0.1413	and	0.0522
with	0.0432	of	0.0288	thy	0.0658	a	0.0515	of	0.0354
doth	0.0415	and	0.0277	in	0.0611	love	0.0436	her	0.0262
and	0.0293	with	0.0273	is	0.0433	heart	0.0194	that	0.0228
shall	0.0222	is	0.0261	your	0.0341	so	0.0175	me	0.0216
in	0.0196	thou	0.0219	all	0.0221	this	0.0171	for	0.0188
which	0.0162	world	0.0157	that	0.0203	self	0.0156	thee	0.0138
may	0.0119	so	0.0131	as	0.0198	beauty	0.0150	eyes	0.0134
will	0.0102	by	0.0130	of	0.0176	their	0.0111	the	0.0129
did	0.0102	more	0.0130	sweet	0.0159	me	0.0108	fair	0.0122

State 5		State 6		State 7		State 8		State 9	
Word	Prob	Word	Prob	Word	Prob	Word	Prob	Word	Prob
of	0.1040	i	0.1001	but	0.1460	and	0.1873	i	0.0649
be	0.0461	thou	0.0582	and	0.1119	that	0.0950	not	0.0355
her	0.0425	in	0.0498	so	0.0744	when	0.0657	do	0.0313
my	0.0374	all	0.0336	then	0.0585	as	0.0437	that	0.0207
to	0.0325	to	0.0297	for	0.0546	but	0.0429	can	0.0204
me	0.0265	she	0.0297	o	0.0402	or	0.0374	more	0.0149
thee	0.0256	that	0.0282	what	0.0365	for	0.0359	it	0.0137
his	0.0245	you	0.0227	yet	0.0340	if	0.0355	with	0.0130
in	0.0223	it	0.0208	how	0.0336	which	0.0262	thou	0.0130
make	0.0214	on	0.0186	which	0.0310	nor	0.0230	doth	0.0126

7 Extra Credit

In addition to rhyme, we have implemented the additional goals of “additional texts” and “meter.” In the additional texts, when we were working on the texts, we realised that Shakespeare’s texts had Hindu-Arabic numbers whereas Amoretti had Roman numbers. There was some difficulty at the start but we were eventually able to successfully get rid of the Roman numbers.

We have already described our second additional goal of stress measures.