Late hours used: 43

1 Introduction

- Team Name: Darth Jar Jar
- Members: Ben Juarez, Dallas Taylor, Kyle McGraw
- Piazza Post: Linked here
- Division of Labor: Ben and Dallas handled most of the pre-processing for the Hidden Markov Models while Kyle handled the additional pre-processing needed for the Recurrent Neural Network implementation. Ben did the majority of the Naive Poem Generation from HMMs, Kyle implemented the RNN, while Dallas also worked on the HMM implementation, particularly with the additional goals. Ben did the Piazza post, and Dallas also generated the visualizations. The report was divided appropriately. Overall, the workload was distributed equally.

2 Pre-Processing

Code

Hidden Markov Models

Overall, we kept most of our pre-processing of *shakespeare.txt* simple for the HMM implementation such that it was in accordance with the implementation from HW6. The function *sonnet_parse_observations* is heavily based off the function *parse_observations* from HW6. The difference is that we filter out lines of size ≤ 1 in order to remove the numbered headers and the empty lines since these would not positively contributed to our modeling. Therefore, each line was split into words such that the words were lower-cased and punctuation/other characters were removed. By looking through *shakespeare.txt*, we made the decision to exclude all punctuation and other special characters since we did not think it would impact our ability to reconstruct sonnets in Shakespeare's voice. We also made the decision to remove all capitalizations in order to keep everything standardized and simple. Although, we did capitalize the beginning of each sentence and certain words like I and O so that we could retain some sense of proper format. One issue that we had with our choices for pre-processing was that we did not split hyphenated words, so our poems could potentially have words that were previously hyphenated but are now merged together with no punctuation. With more time, we would have handled special cases like this.

We also did some pre-processing with *syllable_dictionary.txt* so that we could efficiently and properly count the number of syllables. We essentially created two dictionaries for tracking syllable counts. In order to allow for the proper matching of words, the words in the syllable dictionary were processed in the same way as the previous section such that capitalization was removed along with any punctuation and special characters. Therefore, we created two python dictionaries with these processed words such that one dictionary maps the word to its syllable count while the other dictionary maps the observation number of the word (from *obs_map*) to the syllable count. We did this since the first dictionary (*syllab_dict*) was used to print out the number of syllables in each line while the second dictionary (*syllab_dict2*) was utilized in *sonnet_generate_emission* to generate emissions based on number of syllables (this process will be explained in further detail later in this report). An important note with the generation of these dictionaries was that we maintained a one to one mapping of word to its number of syllables. For the words with end-of-line syllable counts in addition to their real syllable count, we always arbitrarily selected the syllable count that appeared first and disregarded the other count. We made this decision in order to maintain simplicity and efficiency with other functions. We also did this because there were only a handful of words that had more than one syllable count, so we did not think this would have a major effect on poem generation.

We then revisited *shakespeare.txt* so that we could introduce rhyme schemes to our poems. We similarly created two dictionaries for tracking which words rhyme with which words. In order to allow for the proper matching of words, the words in the rhyme dictionaries were processed in the same way as the previous sections such that capitalization, punctuation, and special characters are removed. Therefore, we created two python dictionaries with these processed words such that one dictionary maps from the word (string) to a list of words (strings) while the other dictionary maps from the observation number of the word (from *obs_map*) to a list of words (strings). We did this since the first dictionary (*rhyme_dict*) was used to facilitate the comparison between rhyming words for debugging and direct string word access while the second dictionary (*rhyme_dict2*) was utilized in *rhyme_generate_emission* to generate emissions with a required rhyming word (explained later in the report). An important note with the generation of these dictionaries was that we check for a desired number of rhyming words when we want to create a specific rhyming scheme, and thus we always guarantee that we have enough rhyming words for each scheme. This is directly implemented both within *rhyme_generate_emission* and *rhyme_sample_sentences*. We made the decision to utilize dictionaries for rhyming in this nature so that we could easily access a list of words that rhyme with a given word (that does not contain the word itself).

Recurrent Neural Networks

Our pre-processing for the RNN was done very similarly to that for the HMMs. For our RNN, we decided to still remove capitalization, but decided to keep some punctuation including new lines, commas, apostrophes, periods, exclamation points, and question marks. We did this because the seed includes these punctuation and we expected that for the character-based RNN that they would not have a large effect (should be able to learn punctuation same as the other characters). Our generated training data consists of semi-redundant (every 3rd character), 40-character sequences from the original dataset with each character encoded using one-hot encoding.

3 Unsupervised Learning

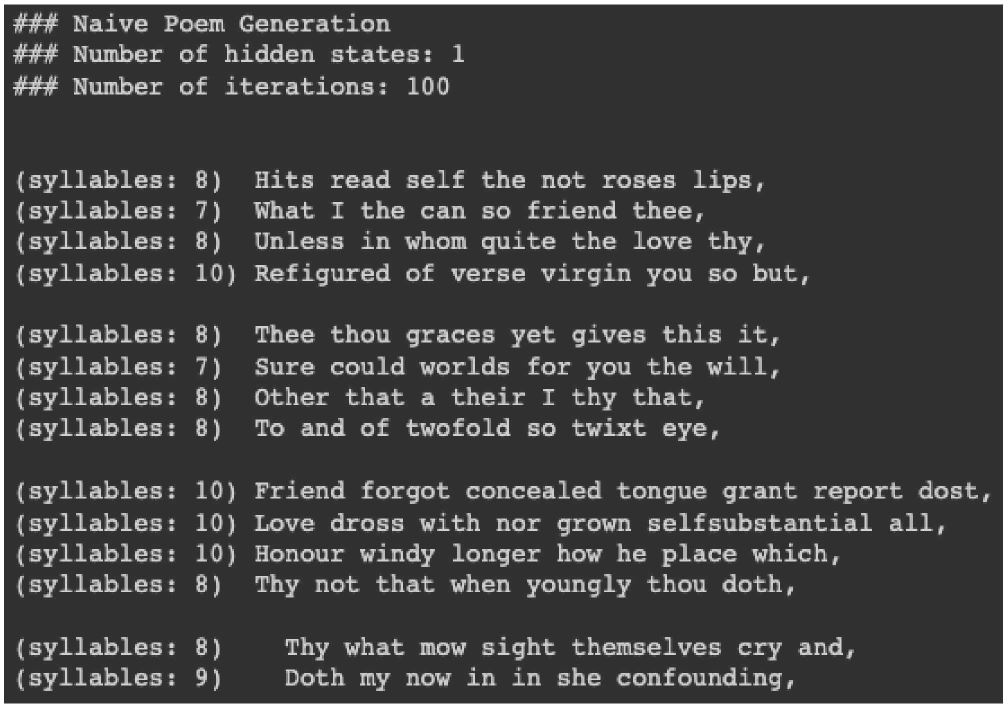
Code

As directed, we used the Baum-Welch algorithm that was implemented in HW6. We did not use any additional packages besides what was used in HW6. As investigated in HW6, we suspected that increasing the number of hidden states would result in phrases that make more sense. Therefore, we experimented with the number of hidden states varying from 1 to 15. As expected, it seemed like the models with more hidden states produced better sounding poems. Thus, for quick testing we used smaller numbers of hidden states since the run-time was shorter, but we mainly produced poems with models with 15 hidden states. Generally, we also almost always used 100 epochs. As a final note, we only considered models with 15 hidden states or fewer due to time efficiency.

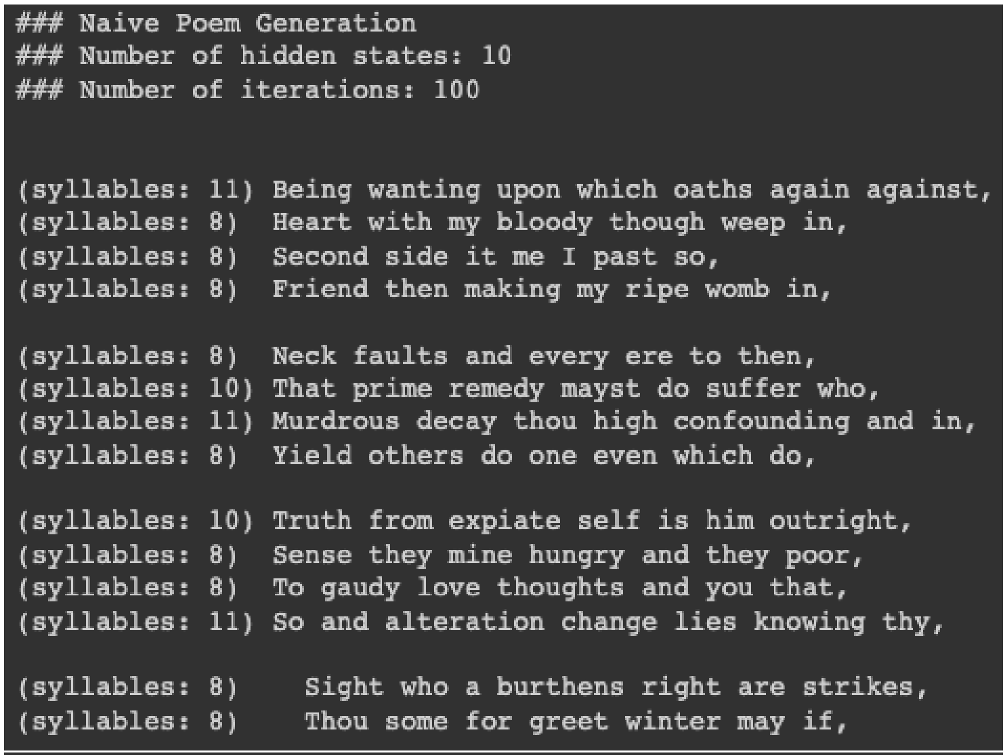
4 Poetry Generation, Part 1: Hidden Markov Models

Code

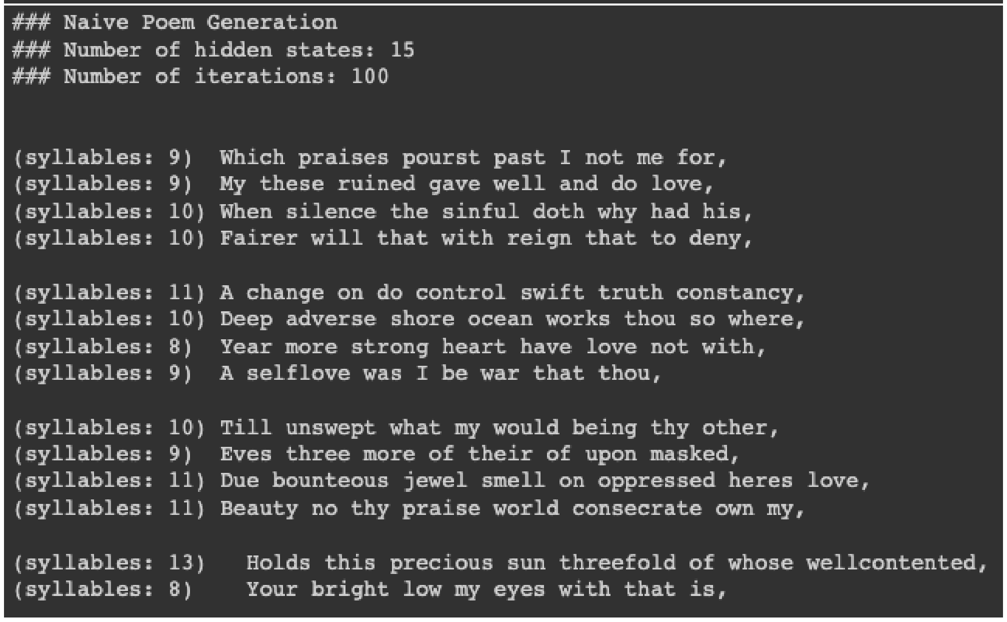
Let us describe our process for Naive Poem Generation for a 14-line sonnet (not controlling for exact syllable count, rhyme, etc). We followed a very similar process that was seen in HW6. Using *obs* generated from our *sonnet_parse_observations* function (essentially equivalent to HW6 *parse_observations*), we trained our unsupervised HMM appropriately. The process of generating a sonnet from this trained HMM comes from our *generate_naive_sonnet* function. At its core, this function generates the 14 lines of poetry individually using *sample_sentence* (from HW6) along with handling the proper splitting into quartains/couplets. in other words, *sample_sentence* was called 14 times in this naive poem generation process. We also counted the syllables in each line after the sentences have been generated, and also capitalized words like I and O. Within *sample_sentence*, we are able to retrieve the words from the list of emissions from *generate_emission* (also from HW6). Since this function accounts for number of words rather than number of syllables for each line, we attempted to get as close to 10 syllables per line as possible by randomly choosing between 7 or 8 words each line. Let us note that proper syllable count was implemented as a part of the Additional Goals section. The following poems were (naively) generated with this algorithm (hidden states: 1, 10, 15):



```
### Naive Poem Generation
### Number of hidden states: 1
### Number of iterations: 100


(syllables: 8)  Hits read self the not roses lips,
(syllables: 7)  What I the can so friend thee,
(syllables: 8)  Unless in whom quite the love thy,
(syllables: 10) Refigured of verse virgin you so but,

(syllables: 8)  Thee thou graces yet gives this it,
(syllables: 7)  Sure could worlds for you the will,
(syllables: 8)  Other that a their I thy that,
(syllables: 8)  To and of twofold so twixt eye,

(syllables: 10) Friend forgot concealed tongue grant report dost,
(syllables: 10) Love dross with nor grown selfsubstantial all,
(syllables: 10) Honour windy longer how he place which,
(syllables: 8)  Thy not that when youngly thou doth,

(syllables: 8)    Thy what mow sight themselves cry and,
(syllables: 9)    Doth my now in in she confounding,
```



```
### Naive Poem Generation
### Number of hidden states: 10
### Number of iterations: 100


(syllables: 11) Being wanting upon which oaths again against,
(syllables: 8)  Heart with my bloody though weep in,
(syllables: 8)  Second side it me I past so,
(syllables: 8)  Friend then making my ripe womb in,

(syllables: 8)  Neck faults and every ere to then,
(syllables: 10) That prime remedy mayst do suffer who,
(syllables: 11) Murdrous decay thou high confounding and in,
(syllables: 8)  Yield others do one even which do,

(syllables: 10) Truth from expiate self is him outright,
(syllables: 8)  Sense they mine hungry and they poor,
(syllables: 8)  To gaudy love thoughts and you that,
(syllables: 11) So and alteration change lies knowing thy,

(syllables: 8)    Sight who a burthens right are strikes,
(syllables: 8)    Thou some for greet winter may if,
```



```
### Naive Poem Generation
### Number of hidden states: 15
### Number of iterations: 100


(syllables: 9)  Which praises pourst past I not me for,
(syllables: 9)  My these ruined gave well and do love,
(syllables: 10) When silence the sinful doth why had his,
(syllables: 10) Fairer will that with reign that to deny,

(syllables: 11) A change on do control swift truth constancy,
(syllables: 10) Deep adverse shore ocean works thou so where,
(syllables: 8)  Year more strong heart have love not with,
(syllables: 9)  A selflove was I be war that thou,

(syllables: 10) Till unswept what my would being thy other,
(syllables: 9)  Eves three more of their of upon masked,
(syllables: 11) Due bounteous jewel smell on oppressed heres love,
(syllables: 11) Beauty no thy praise world consecrate own my,

(syllables: 13)   Holds this precious sun threefold of whose wellcontented,
(syllables: 8)    Your bright low my eyes with that is,
```

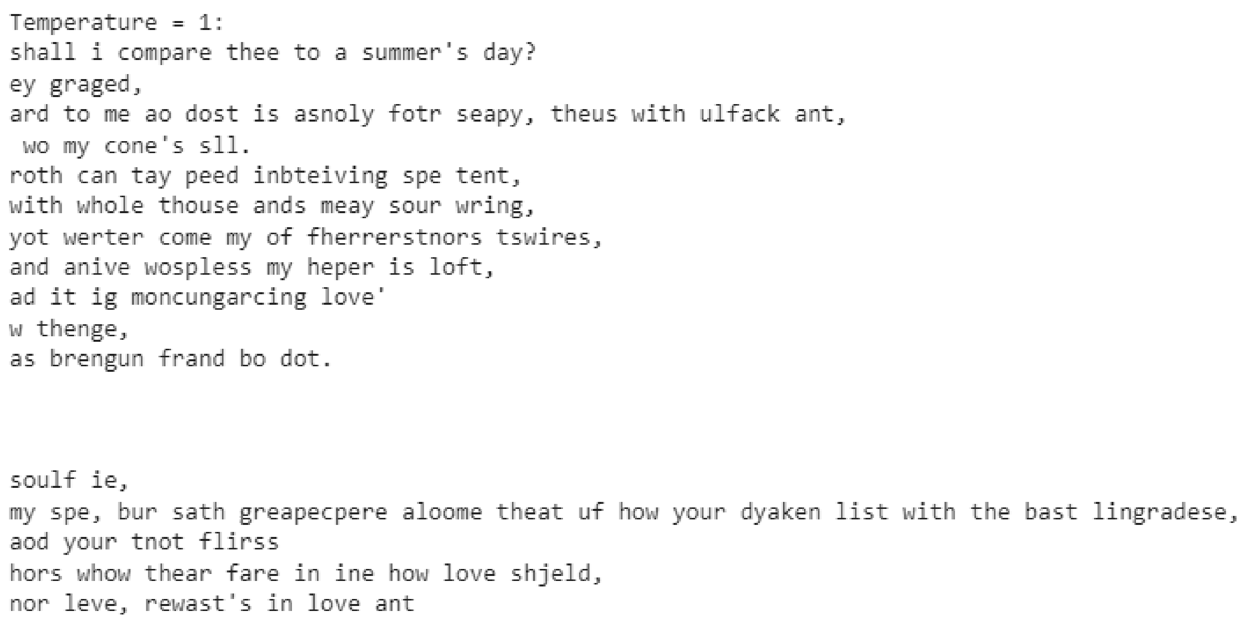
Examining these poems, we can see that they don't really make much sense. There is not rhyming pattern. The syllable count, as displayed, is somewhat close to 10 syllables for each line. This is purely due to either having 7 or 8 words for each line. This helps to retain some level of structure and rhythm, but it is not very consistent. However, it does seem like these poems express Shakespeare's voice generally in the sense that the words used are not quite modern. Although these poems do not make much sense, they still seem to retain remnants of Shakespeare's voice. Furthermore, it seems like increasing the number of hidden states (up to 15) helps to generate lines of poetry that are somewhat more realistic and make more sense such that they seem to better resemble lines from Shakespeare's sonnets. This makes sense because with more hidden states, we are able to better retain proper sentence structure as opposed to randomly places words which is the case with 1 hidden state. Thus, with more hidden states, we would likely see related words in each of the different states which would help to retain phrases that are more accurate to what we saw in Shakespeare's actual sonnets. Overall, generating poems in this naive manner is not very effective, but it still interesting to try and read/analyze the produced poems. We will see improvements to this process with the Additional Goals.

5 Poetry Generation, Part 2: Recurrent Neural Networks

Code

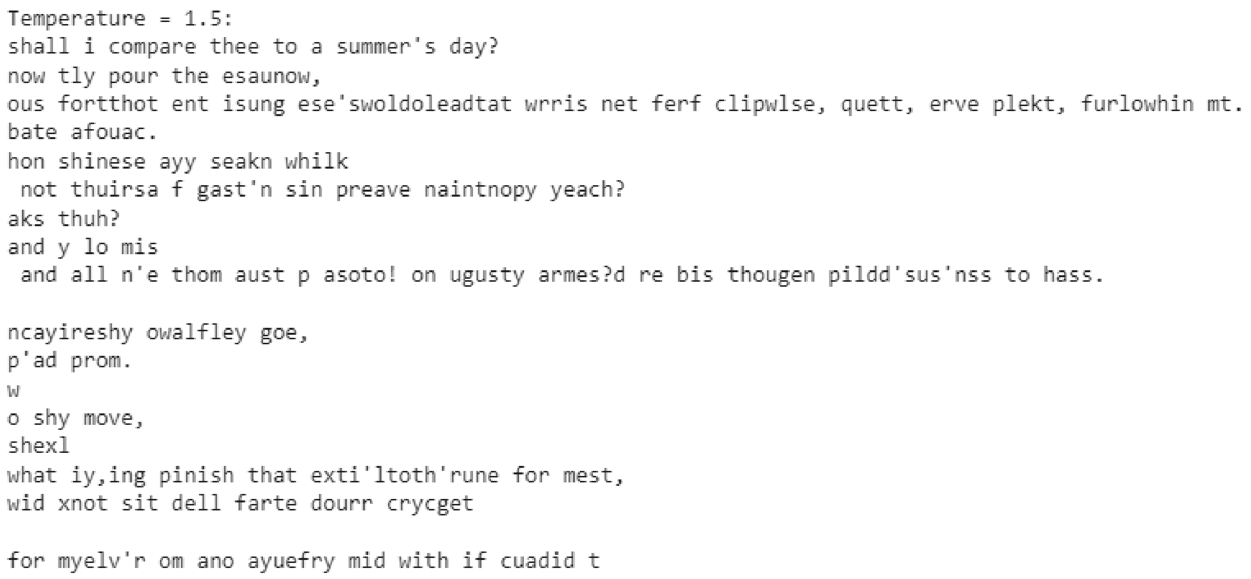
For the RNN, we used Pytorch to implement a model with a single LSTM layer of 150 units connected to a fully-connected output layer with a softmax nonlinearity (constructed using Pytorch Module, LSTM, Linear, LogSoftmax). The model was trained for 100 epochs, which we found was sufficient for the loss to converge, to minimize cross-entropy loss (Pytorch CrossEntropyLoss with Adam optimizer and 0.01 learning rate). The model was trained on semi-redundant sequences of 40 characters drawn from the Shakespeare text. 560 character poems were generated by drawing softmax samples from the trained model with varying temperature parameters (1, 1.5, 0.75, and 0.25) using the seed "shall i compare thee to a summer's day?\n":

Temperature = 1:



```
Temperature = 1:
shall i compare thee to a summer's day?
ey graged,
ard to me ao dost is asnoly fotr seapy, theus with ulfack ant,
 wo my cone's sll.
roth can tay peed inbteiving spe tent,
with whole thouse ands meay sour wring,
yot werter come my of fherrerstnors tswires,
and anive wospless my heper is loft,
ad it ig moncungarcing love'
w thenge,
as brengun frand bo dot.



soulf ie,
my spe, bur sath greapecpere aloome theat uf how your dyaken list with the bast lingradese,
aod your tnot flirss
hors whow thear fare in ine how love shjeld,
nor leve, rewast's in love ant
```

Temperature = 1.5:



```
Temperature = 1.5:
shall i compare thee to a summer's day?
now tly pour the esaunow,
ous fortthot ent isung ese'swoldoleadtat wrris net ferf clipwlse, quett, erve plekt, furlowhin mt.
bate afouac.
hon shinese ayy seakn whilk
 not thuirsa f gast'n sin preave naintnopy yeach?
aks thuh?
and y lo mis
 and all n'e thom aust p asoto! on ugusty armes?d re bis thougen pildd'sus'nss to hass.

ncayireshy owalfley goe,
p'ad prom.
w
o shy move,
shexl
what iy,ing pinish that exti'ltoth'rune for mest,
wid xnot sit dell farte dourr crycget

for myelv'r om ano ayuefry mid with if cuadid t
```

Temperature = 0.75 (lines wrap around):

shall i compare thee to a summer's day?
why aroun the werth then is thy burting sue ast ars cleatry frit then sinl'er for me prist,
so chauls whel co ples not the bechance
thing love thee the erencing consted ar wince boows that with praings,
on se me tak in thean to not ay thy wery make pow.
a sear thour with now be thace at thy with that of to stang groued,
theti strow,
do cove,
and thou wat to deak in asing hast will my semered with heart hith whore thon thee thet recaint be dost the sood ther, whon with in my told and i soog in haok love sweet to le

Temperature = 0.25 (lines wrap around):

shall i compare thee to a summer's day?
or thy self will the self the self and the self and the sore thou the self the sould dist and the self the stored the self and the sull the self in the self in the seave thee my self the store the swall my sould the self of thee, the sall the self be the self and dear and the stall when shall heart the self i some the reme,
the with self the with the conder senfor shee,
the the may the have the self i stould dost the self and the self the self i love the praing the make the self the sore the stould thee are the st

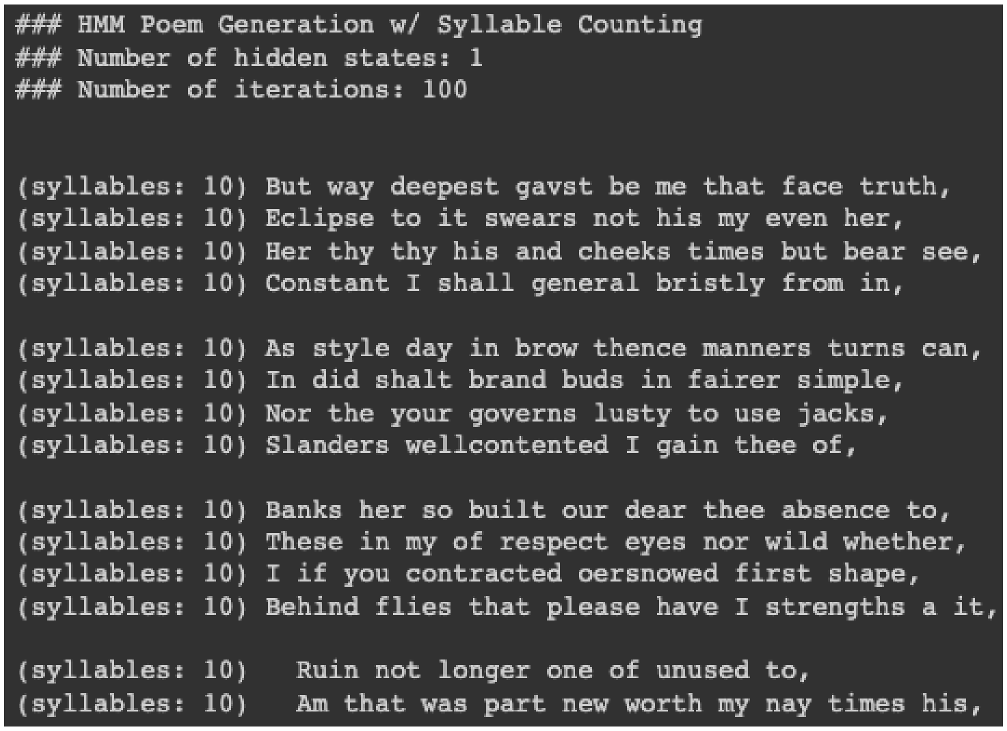
For the different temperatures we see differing levels of correctness vs repetition. The higher temperature poems generate a wide variety of words at the expense of some of these not being real words, while the lower temperature poems generate correct words but has more repetition of words. While the LSTM does successfully learn some of the structure of the sonnets, the poem quality is not as good as those produced by the HMM as well as the RNN requires more training data and a longer runtime than the HMM.

6 Additional Goals

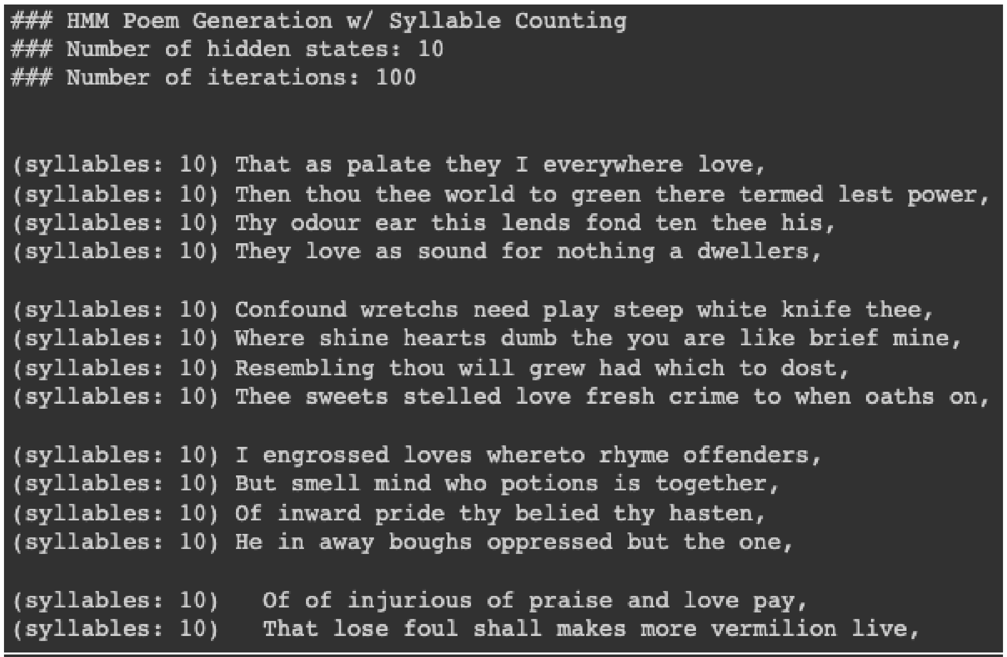
Code

Meter - Counting Syllables

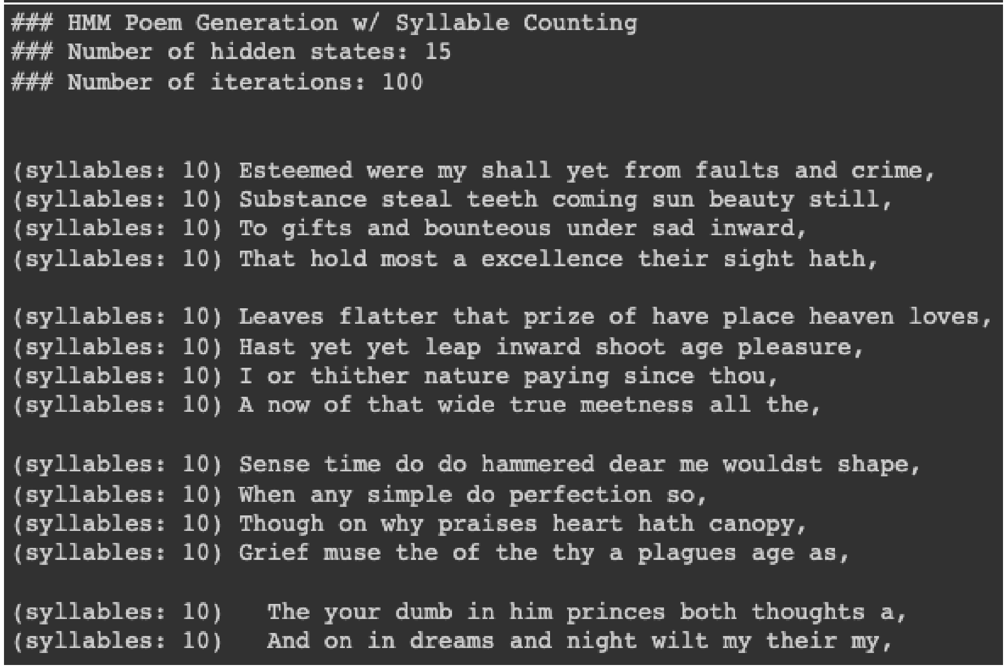
One major improvement to our poem generation with HMMs was incorporating controlled syllable count for each line by labelling each word with its number of syllables. As mentioned, our naive implementation disregarded exact syllable count since it generated lines of poetry based on the number of words. We were able to generate lines of poetry with exactly 10 syllables by altering some of our functions. The key functions involved with this were *generate_sonnet_syllab_count*, *sonnet_sample_sentence*, *sonnet_generate_emission*. As stated in the Pre-Processing section, we produced a dictionary that maps the observation number of each word to its syllable count. Our *sonnet_generate_emission* takes in this dictionary as an argument which generate emissions based on another parameter *n_syllab*. This process is similar to *generate_emission* except that instead of producing emissions based on the length of the sequence (number of words), it produces emissions by tracking the syllable count. Thus, observations are only added to the emission if the max syllable count is not crossed. Therefore, the major change relative to the naive process comes from *sonnet_generate_emission*. The following poems were generated with *generate_sonnet_syllab_count* (generates poems line by line) on trained HMMs with the number of hidden states varying from 1 to 15:



```
### HMM Poem Generation w/ Syllable Counting
### Number of hidden states: 1
### Number of iterations: 100


(syllables: 10) But way deepest gavst be me that face truth,
(syllables: 10) Eclipse to it swears not his my even her,
(syllables: 10) Her thy thy his and cheeks times but bear see,
(syllables: 10) Constant I shall general bristly from in,

(syllables: 10) As style day in brow thence manners turns can,
(syllables: 10) In did shalt brand buds in fairer simple,
(syllables: 10) Nor the your governs lusty to use jacks,
(syllables: 10) Slanders wellcontented I gain thee of,

(syllables: 10) Banks her so built our dear thee absence to,
(syllables: 10) These in my of respect eyes nor wild whether,
(syllables: 10) I if you contracted oersnowed first shape,
(syllables: 10) Behind flies that please have I strengths a it,

(syllables: 10)   Ruin not longer one of unused to,
(syllables: 10)   Am that was part new worth my nay times his,
```



```
### HMM Poem Generation w/ Syllable Counting
### Number of hidden states: 10
### Number of iterations: 100


(syllables: 10) That as palate they I everywhere love,
(syllables: 10) Then thou thee world to green there termed lest power,
(syllables: 10) Thy odour ear this lends fond ten thee his,
(syllables: 10) They love as sound for nothing a dwellers,

(syllables: 10) Confound wretchs need play steep white knife thee,
(syllables: 10) Where shine hearts dumb the you are like brief mine,
(syllables: 10) Resembling thou will grew had which to dost,
(syllables: 10) Thee sweets stelled love fresh crime to when oaths on,

(syllables: 10) I engrossed loves whereto rhyme offenders,
(syllables: 10) But smell mind who potions is together,
(syllables: 10) Of inward pride thy belied thy hasten,
(syllables: 10) He in away boughs oppressed but the one,

(syllables: 10)   Of of injurious of praise and love pay,
(syllables: 10)   That lose foul shall makes more vermilion live,
```

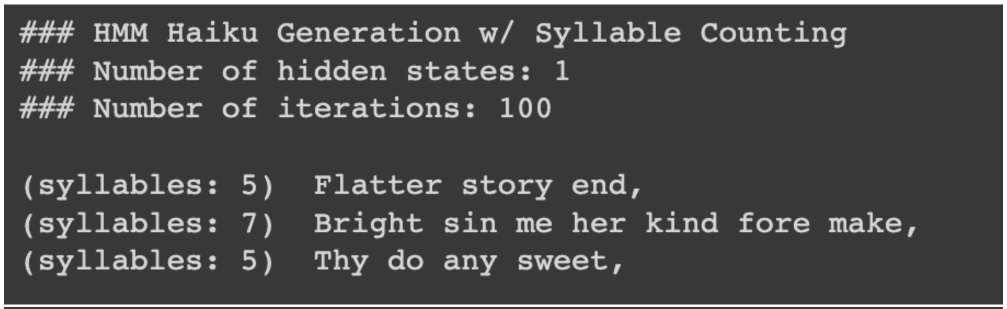


```
### HMM Poem Generation w/ Syllable Counting
### Number of hidden states: 15
### Number of iterations: 100


(syllables: 10) Esteemed were my shall yet from faults and crime,
(syllables: 10) Substance steal teeth coming sun beauty still,
(syllables: 10) To gifts and bounteous under sad inward,
(syllables: 10) That hold most a excellence their sight hath,

(syllables: 10) Leaves flatter that prize of have place heaven loves,
(syllables: 10) Hast yet yet leap inward shoot age pleasure,
(syllables: 10) I or thither nature paying since thou,
(syllables: 10) A now of that wide true meetness all the,

(syllables: 10) Sense time do do hammered dear me wouldst shape,
(syllables: 10) When any simple do perfection so,
(syllables: 10) Though on why praises heart hath canopy,
(syllables: 10) Grief muse the of the thy a plagues age as,

(syllables: 10)   The your dumb in him princes both thoughts a,
(syllables: 10)   And on in dreams and night wilt my their my,
```

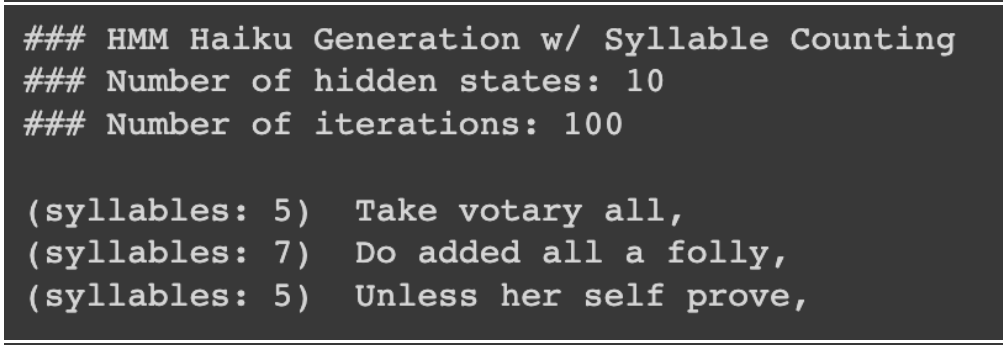
Examining these poems, there doesn't seem to be an incredibly clear difference relative to the naively generated poems in regards to how much the poems actually make sense. However, it does seem that these poems with precise syllable counts have a better rhythm (easier to read). This helps to retain more of Shakespeare's sonnet voice/format. We did not have the time to fully handle word stresses, but accounting for syllable counts in a supervised fashion brings us closer to true iambic pentameter.

Generating Other Poetic Forms - Haiku

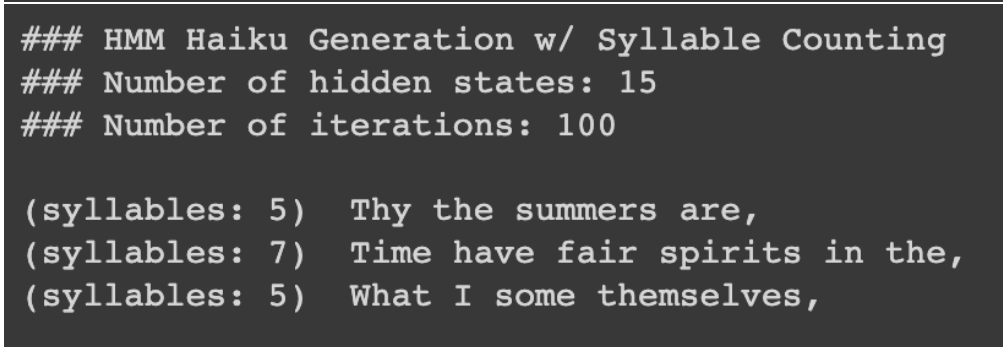
Another improvement to our poem generation with HMMs was producing a different poetic form, specifically a Haiku. This was implemented by slightly altering our syllable counting functionality such that we could specify different syllables for each line. The rest of the functionality is the same as for that discussed in the previous section (Counting Syllables). Thus, we were able to generate lines of poetry with a desired number of syllables for each line by slightly altering some of our funtions. The key functions involved with this were *generate_haiku_syllab_count* and *haiku_sample_sentence*. We utilized the syllable dictionary defined in our Pre-Processing section once again. We still utilize *sonnet_generate_emission* for our haiku generation, and simply change the *n_syllab* parameter as we generate each sentence. Therefore, the major change relative to the Counting Syllables process comes from *generate_haiku_syllab_count* and *haiku_sample_sentence*. The following poems were generated with *generate_haiku_syllab_count* (generates poems line by line) on trained HMMs with the number of hiddens tates varying from 1 to 15:



```
### HMM Haiku Generation w/ Syllable Counting
### Number of hidden states: 1
### Number of iterations: 100

(syllables: 5)  Flatter story end,
(syllables: 7)  Bright sin me her kind fore make,
(syllables: 5)  Thy do any sweet,
```



```
### HMM Haiku Generation w/ Syllable Counting
### Number of hidden states: 10
### Number of iterations: 100

(syllables: 5)  Take votary all,
(syllables: 7)  Do added all a folly,
(syllables: 5)  Unless her self prove,
```



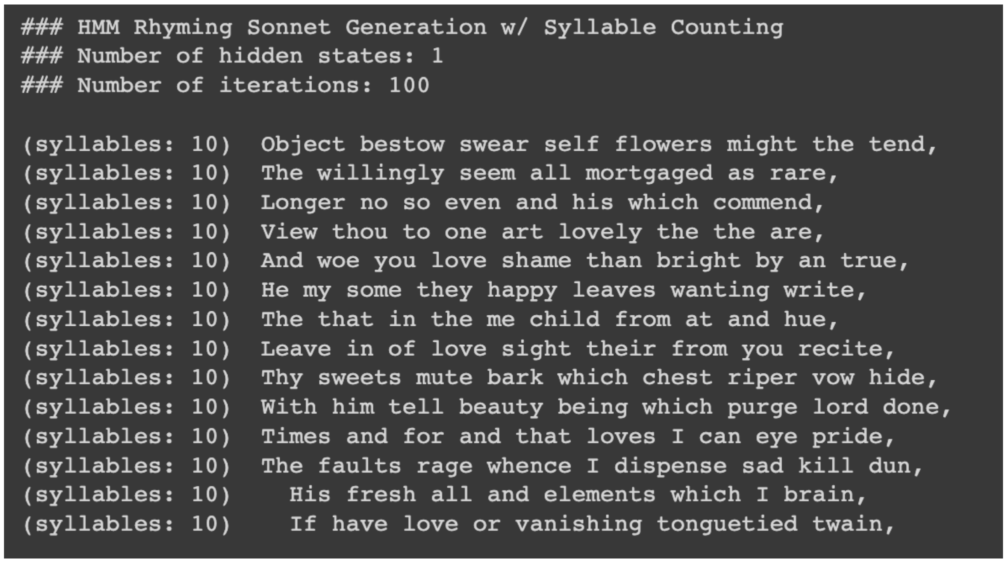
```
### HMM Haiku Generation w/ Syllable Counting
### Number of hidden states: 15
### Number of iterations: 100

(syllables: 5)  Thy the summers are,
(syllables: 7)  Time have fair spirits in the,
(syllables: 5)  What I some themselves,
```

Examining these poems, there is still not a change relative to the naively generated poems in regards to how much the poems actually make sense, as would be expected from what changes were made to our functions.

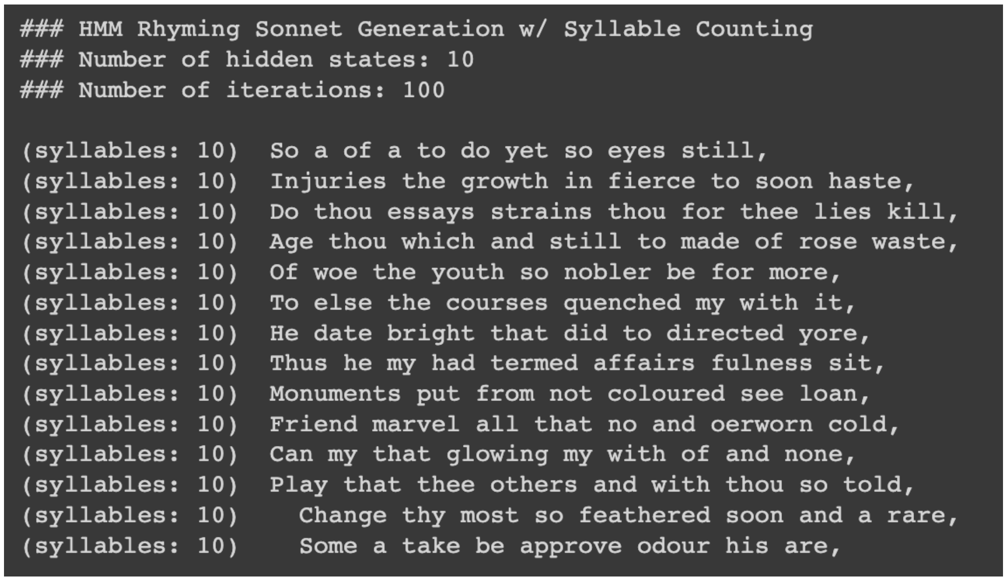
However, it is clear that we have created a haiku with the correct syllable requirements of $5-7-5$. Thus, we are able to generate a different poem form, as desired.

Rhyme

A final large improvement to our poem generation with HMMs was introducing the ability to produce a rhyme scheme within our poems. This was introduced by selecting a set of rhyming words from a dictionary and building each line backwards from each rhyming word. We additionally retained the syllable count control in our implementation. The key functions involved with this were *generate_rhyming_haiku_syllab_count*, *generate_rhyming_sonnet_syllab_count*, *rhyme_sample_sentences*, and *rhyme_generate_emission*. As stated in the Pre-Processing section, we produced a dictionary that maps the observation number of each word to a list of string words that were given to rhyme with that word. Our *rhyme_generate_emission* takes in this dictionary as an argument (as well as the syllable dictionary mentioned in the Meter section above) which generate emissions based on the state that produced the original rhyming word, the number of syllables in that rhyming word, and total *n_syllab*. This process is extremely similar to *sonnet_generate_emission* and *generate_emission* except that we now produce emissions by tracking the remaining syllable count produced by related words of our rhyming word that are added to our sentence. Thus, observations are only added to the emission if the max syllable count is not crossed and the first word and corresponding state is from a predetermined rhyming word. We can additionally see that when we generate our overall poem, we have to construct our entire poem before hand, by calling *rhyme_sample_sentences* (and *haiku_sample_sentence* if making a rhyming haiku) for each individual rhyme scheme and then constructing the poem by selecting the corresponding lines of each rhyme scheme, where we can then correct capitalization and spacing while visualizing syllable counts. Therefore, the major change relative to the naive process and counting syllables process comes from BOTH *rhyme_generate_emission* and *generate_rhyme_haiku_syllab_count*/*generate_rhyme_sonnet_syllab_count*. The following poems were generated with *generate_rhyming_sonnet_syllab_count* (generates poems rhyme scheme group by rhyme scheme group and then line by line within each group) on trained HMMs with the number of hidden states varying from 1 to 10:

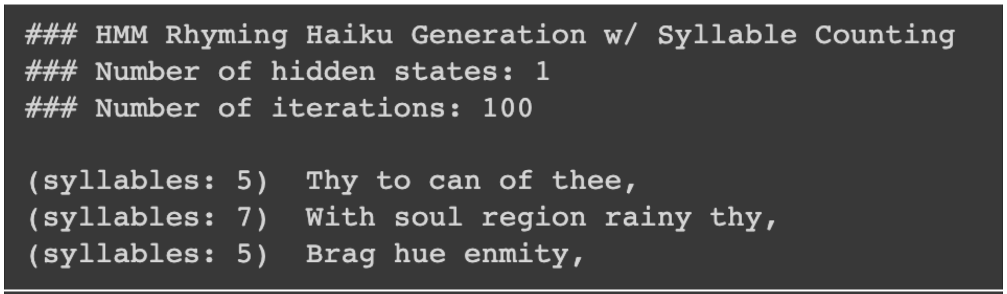


```
### HMM Rhyming Sonnet Generation w/ Syllable Counting
### Number of hidden states: 1
### Number of iterations: 100

(syllables: 10)  Object bestow swear self flowers might the tend,
(syllables: 10)  The willingly seem all mortgaged as rare,
(syllables: 10)  Longer no so even and his which commend,
(syllables: 10)  View thou to one art lovely the the are,
(syllables: 10)  And woe you love shame than bright by an true,
(syllables: 10)  He my some they happy leaves wanting write,
(syllables: 10)  The that in the me child from at and hue,
(syllables: 10)  Leave in of love sight their from you recite,
(syllables: 10)  Thy sweets mute bark which chest riper vow hide,
(syllables: 10)  With him tell beauty being which purge lord done,
(syllables: 10)  Times and for and that loves I can eye pride,
(syllables: 10)  The faults rage whence I dispense sad kill dun,
(syllables: 10)    His fresh all and elements which I brain,
(syllables: 10)    If have love or vanishing tonguetied twain,
```

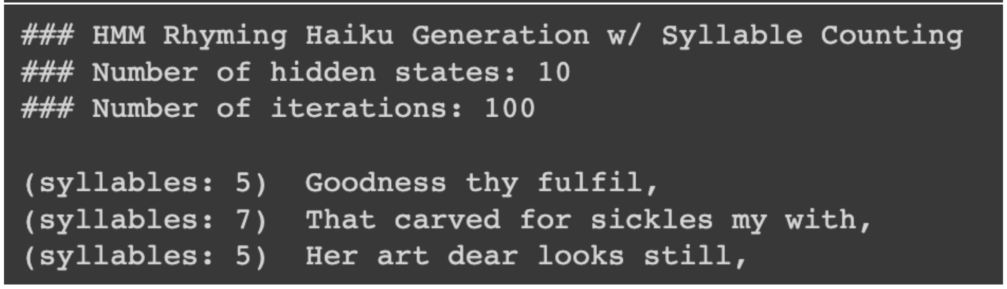


```
### HMM Rhyming Sonnet Generation w/ Syllable Counting
### Number of hidden states: 10
### Number of iterations: 100

(syllables: 10)  So a of a to do yet so eyes still,
(syllables: 10)  Injuries the growth in fierce to soon haste,
(syllables: 10)  Do thou essays strains thou for thee lies kill,
(syllables: 10)  Age thou which and still to made of rose waste,
(syllables: 10)  Of woe the youth so nobler be for more,
(syllables: 10)  To else the courses quenched my with it,
(syllables: 10)  He date bright that did to directed yore,
(syllables: 10)  Thus he my had termed affairs fulness sit,
(syllables: 10)  Monuments put from not coloured see loan,
(syllables: 10)  Friend marvel all that no and oerworn cold,
(syllables: 10)  Can my that glowing my with of and none,
(syllables: 10)  Play that thee others and with thou so told,
(syllables: 10)    Change thy most so feathered soon and a rare,
(syllables: 10)    Some a take be approve odour his are,
```

The following poems were generated with *generate_rhyming_haiku_syllab_count* (generates poems rhyme scheme group by rhyme scheme group and then line by line within each group) on trained HMMs with the number of hidden states varying from 1 to 10:



```
### HMM Rhyming Haiku Generation w/ Syllable Counting
### Number of hidden states: 1
### Number of iterations: 100

(syllables: 5)  Thy to can of thee,
(syllables: 7)  With soul region rainy thy,
(syllables: 5)  Brag hue enmity,
```



```
### HMM Rhyming Haiku Generation w/ Syllable Counting
### Number of hidden states: 10
### Number of iterations: 100

(syllables: 5)  Goodness thy fulfil,
(syllables: 7)  That carved for sickles my with,
(syllables: 5)  Her art dear looks still,
```

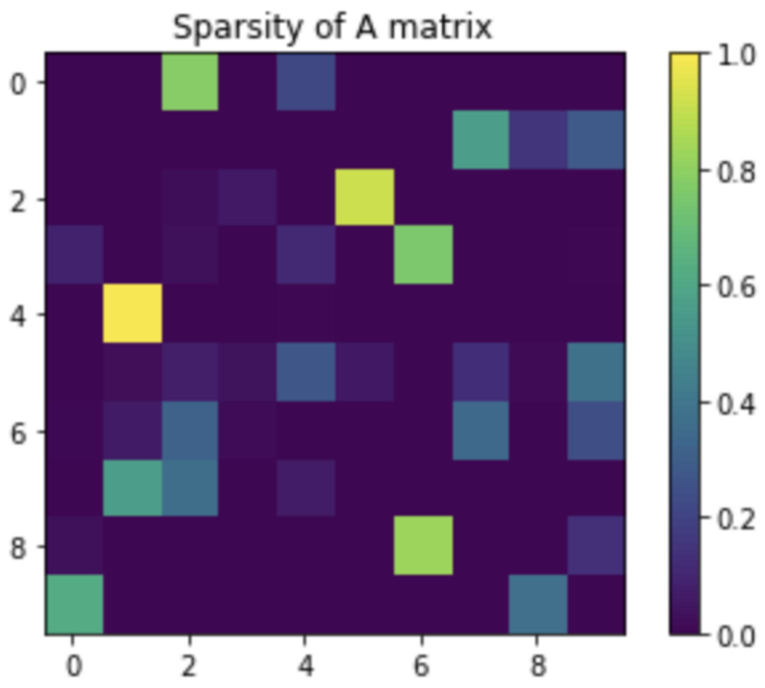
Examining these poems, there is still not a change relative to the naively generated poems in regards to how much they make sense, as expected. However, it is clear that we have successfully allowed for the implementation of various rhyme schemes when generating our poems. Additionally, the rhyme schemes of the poem help to maintain more of a poetic nature, and match Shakespeare's format of poem even closer.

7 Visualization and Interpretation

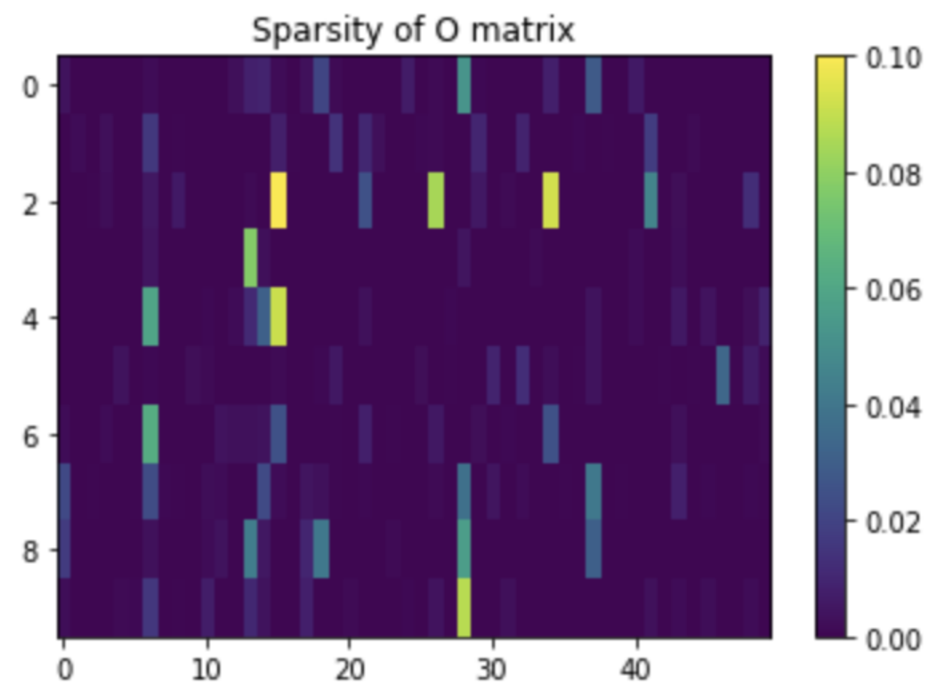
Code

HMM (10 hidden states) Visualizations

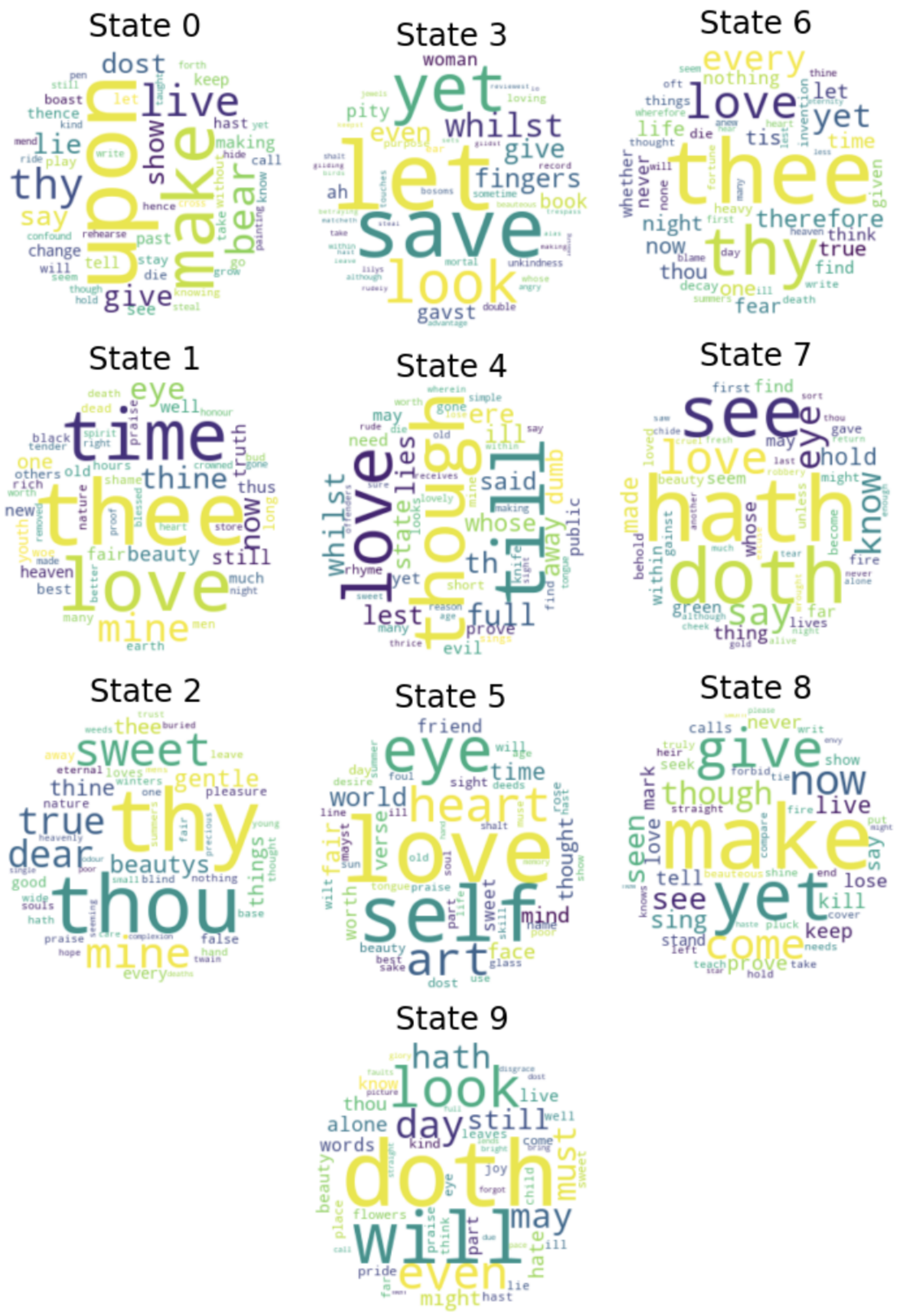
- Sparsity of A matrix:



- Sparsity of O matrix:



- State Wordclouds:



Discussion

When examining the sparsity of our A and O matrices, we see that a majority of the values are close to 0, similar to that of previous assignments. We can thus similarly determine that the few non-zero entries in our A matrix indicate a strong transition ”probability” from certain states to certain states (such as $A_{[4,1]}$). Also, the non-zero entries in our O matrix indicate a strong output ”probability” of various emissions. The sparsities of each matrix are on a similar level, with matrix O perhaps being slightly less sparse (meaning that there is a stronger connection between state and outcome vs state and state).

When examining the wordclouds of each state, we can notice the following for each state:

- State 0: Seems like these words represent verb subjects or various words that connect various parts of sentences. This makes sense since state 0 does not produce a lot of emissions or have many other state-state connections and is thus rarely visited.
- State 1: Seems like these words represent objects or specifically objects of beauty. This makes sense since transitioning from state 1 does not result in many emissions or states, but there is a high probability of transitioning TO state 1.
- State 2: Similarly seems like these words represent objects or specifically objects of beauty.
- State 3: Seems like these words represent articles or sentence interruptions (whilst, let, yet, save, even).
- State 4: Seems like these words represent adjectives, verbs and adverbs (though, till, said, lies, prove).
- State 5: Seems like these words also represent objects or specifically objects of beauty.
- State 6: Seems like these words also represent articles or sentence interruptions (thee, thy, therefore, every, yet).
- State 7: Seems like these words represent mostly verbs (see, hath, doth, know, say).
- State 8: Seems like these words also represent articles or sentence interruptions (thee, thy, therefore, every, yet).
- State 9: Seems like these words also represent articles or sentence interruptions (thee, thy, therefore, every, yet).

Additionally interesting in how we see that certain states correspond with various types of words and that those word states also connect with desirable types of word states. We can also see that certain words are present in a lot of states, such as *love* and *doth*, meaning that they are commonly present in a lot of poem sentences and can play multiple roles in a sentence.