Miniproject 3: Poetry Generation

Introduction

Group Members:

Eli Pinkus
Akshay Vegesna

Group Name

Yasser's Disciples

Division of Labor

Eli spent the majority of his time working on HMM models and various refinements to the that model.

Akshay spent the majority of his time working on neural network approaches and dealing with the numerous challenges brought about by a character based approach.

Altogether work was well split and effort was evenly distributed.

Preprocessing

Hidden Markov Models

Data pre-processing for HMM without any rhyming or other advanced considerations was fairly straightforward. We read in the shakespear.txt file in a jupyter and split each line in to a list of its constituent words using the split method of the python string class. We filtered out lines that had size ≤ 1 as we did not want the sonnet headers nor the empty lines to be considered in the model. We did not concern ourselves with punctuation and other characters as we recalled that our set 6 implementation of HMM filtered these characters where necessary.

After the above processing we recompiled the text into string form and used the resulting string to feed the parse_observations helper function to retrieve the observations and the observations mapping from words to indices.

We also wanted our model to be capable of emitting lines of specific syllabic length in a nontrivial way (i.e. not placing random words at end to satisfy syllable count). This was a two-step process. The first step involved the pre-processing task of compiling a syllable dictionary for the observation in our Markov model. We used the provided Syllable_dictionary.txt file and regular expression manipulation to create a mapping from observation number (a indicator of word), to number of syllables associated with that word. We noted that certain words could take on various syllable values depending on context. In those cases, we chose arbitrarily between the possibilities. We passed the result mapping directly to the HMM object to use in the second step which involves changing the way in which our model generates strings. We will discuss that process in greater depth in part 1.

Recurrent Neural Networks

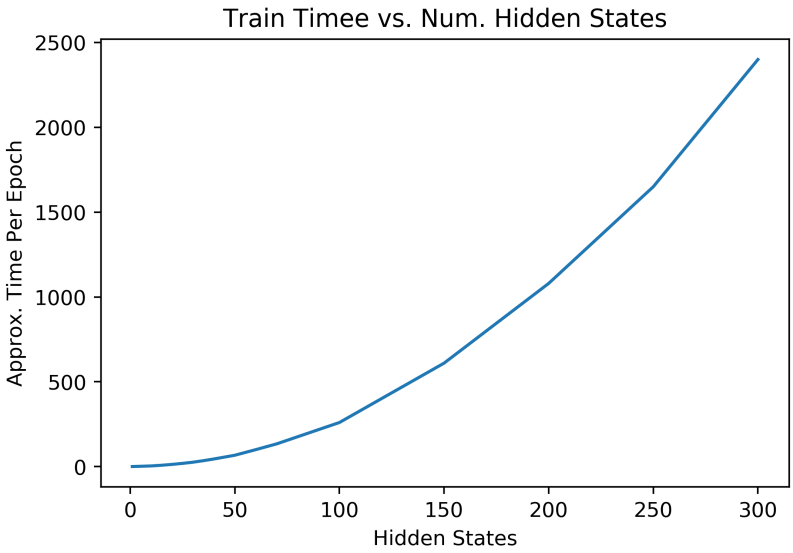
For the recurrent neural network implementation, we tried a few different approaches for pre-processing. Our first approach was to just run the recurrent neural network on sequences on the exact raw data available in the document. This required a dictionary size of 60. We realized soon after that capital letters consisted of 26 letters of this, so we decided to remove capital letters from our implementation, because the model appeared to both converge better and obtain better predictions.

To input our data into the recurrent neural network, we split the dataset into 40 character sequences, buffered by a 5 characters. For each one of these 40 character sequences, we created a 40 by (vocab_size) array representing the character sequence on Keras. This corresponds to having 40 characters with each one having a 1 by (vocab_size) array representing one hot_encodings of what character the sequence was. These encodings were done by having a forward_mapping dictionary which converted characters to integers, and then using keras.utils.to_categorical to convert these integers into one hot-encodings. To go the other way, we used np.argmax to get the index with the maximum value and then used the 'backward_mapping' dictionary to get the character sequence. Once we got this architecture in place, it was a simple matter to pass the data into the Neural Network and train.

Unsupervised Learning

As suggested we used the Baum-Welch algorithm as implemented in the HW6 solutions for unsupervised training. In set 6 we observed that an increase in hidden states of a HMM is usually accompanied by an increase in syntactically correct sentence elements. That being said we also had to deal with the practical consideration of training time. We realized we would have to find a trade-off between model flexibility and training time. We thought it would be useful to gain an understanding of the way in which training time scaled with increasing hidden states. To do so, we timed the training of a single epoch at various hidden states. We did three such timings for each number of hidden states we tested. The results are shown below in table and graph form:

States	1 Epoch Time (s)	Expected 100 epoch time (hr)
1	0.5	0.014
5	2	0.056
10	4	0.11
15	8	0.22
20	13	0.36
25	19	0.53
30	26	0.72
35	35	0.97
40	45	1.3
50	67	1.9
60	100	2.8
70	134	3.7
100	260	7.2



150	610	17
200	1080	30
250	1650	46
300	2400	67

Although there are certainly more sophisticated techniques to analyse complexity, the above data indicates to use that the train time of the model is convex with respect to the number of hidden states. This further emphasizes the importance of finding a practical tradeoff point as train time becomes unfeasible quickly. We figured it would be reasonable to wait about a half an hour for training to 100 epochs in order to maintain a moderate work efficiency so we mostly experimented with 25 hidden states or fewer.

Poetry Generation, Part 1: Hidden Markov Models

Ultimately we had 3 different techniques for sonnet generation from the trained HMM. We started with naïve generation as in set 6, proceeded to control for syllable count and finally considering both rhyme and syllable count. Naïve generation involves outputting 14 lines of 'poetry' based on the trained HMM's transition matrix and observation matrix in an identical manor as in set 6. This method neglected to consider any of the fundamental structures of a sonnet apart from the 14 line feature. As such these poems did not read well and the length of a given line was arbitrary with regards to number of syllables.

The following example was generated by an HMM with 15 hidden states trained for 100 iterations. Each line was randomly chosen to be of either 6,7,8, or 9 words.

```
Heart curse time a me say not defeated...,
Young sullen each for a the rose...,
Thee with cured thou bereft both making resembling...,
Hath or a worth think is...,
That and there days heart show bound but I...,
For things so for overgoes in the violet...,
Poor sing and means thou can...,
Heart well no privilege of it vanished monsters done...,
Today not sufficed their chary as...,
Not saw beautys rose lips given...,
Thou with an state and thy part gentle leaves...,
That I am heavens can mistress vexed art his...,
So moon as o mind I...,
I should gazed truth rain should most runs...,
```

We note that this sonnet has no discernible rhythm or rhyme sceme and only bits and pieces form syntactically correct clauses. It retains Shakespeare's voice in that it incorporates the words from Shakespeare sonnets many of which are considered relics of centuries old English writers and are noticeable relative to modern English terminology. Improvements to this naïve techniques will be discussed in-depth in the additional goals section.

In a qualitative sense, it was clear, through experimentation with various number of hidden states, that a greater number of hidden states results in a greater proportion of syntactically correct emissions. It is possible that too many hidden states would have the model

replicating more so then generating however we cannot afford the computational expense to explore such high numbers of hidden states.

Poetry Generation, Part 2: Recurrent Neural Networks

Our implementation for Recurrent Neural Networks was a simple Keras LSTM network with one layer of 200 LSTM's and one softmax fully connected layer with the number of nodes equal to the vocabulary size. We used a batch size of approximately 20 and trained for 150 epochs to ensure that we got convergence. The loss was approximately 0.1 and the training accuracy was approximately 0.96. We didn't really think about overfitting here since we were chiefly concerned with ensuring that the model learned as much as possible from the data.

The model did learn the sentence and sonnet structure as is clear from the poems generated below, though not all the words were found to be intelligible. The lines all appear to be very close to ten syllables as well. However, the runtime was around 2 hours to obtain sonnets of this quality. For lower training time, we obtained much worse performance—if we only trained for a couple epochs, the model generated the same sentence over and over again, which was off-putting until we realized the convergence behaviour.

We did attempt different pre-processing methods and observed their possible effects on the poetry generation. It turned out that although convergence behaviour was faster, the poems that we obtained after convergence seemed to be of the same quality that we obtained below. This was hard to verify with certainty because of the absence of a real heuristic that we could use to test for quality of poems. We found this to be a general theme with attempting to find poems.

Compared to HMM poems that we generated, the quality of the poems generated in this method was very significantly worse, though it is definitely possible that our RNN method was flawed in some way. Based on our observations however, we generally understood that compared to the train and convergence time of the HMM unsupervised learning method, the RNN with LSTM's didn't appear to give great performance in text generation.

Effect of Temperature

Below are our generated poems using temperatures of 1.5, 0.75, and 0.25 with the following initial 40-character seed: "shall I compare thee to a summer's day?\n".

T=0.25
shall i compare thee to a summer's day?
O the masting that I store teme's galled,
Thas flest love spenes the pice me hid lind.
The grated leart of arunesed conne
Which his with thy woll be delennes from thee.
For swoet's now hip have doth but you dead,
Not thes by ulled anqusing thy piend
Depbownons sien, and thrthals I maked's gragt,
But with the sworth the swarth though my might's sapfor thee.

For that I some, wat not sour evertouth meas from from,
And frest ond past by undeft more the whate,
And for thee as and and confonnes in doot,

T=0.75
shall i compare thee to a summer's day?
O ther him st me brioks no hape with thee,
When thou deem with sweet from utien thes,
That thou hist crisun eis noth ard his of thee,
  And that your sull and and with the sworth,
  And love hime arany lead ne crave love,
But shall I mark well be dele con croped
Astince shes farl me preveated their yrom
Wing manker of I hisepes it fuush a bronter
This icunrt with the worth wo knows
  And thet I brand fell rave I lond rece,
Whill on thy prauth brthing ene reave my de.

T=1.5
shall i compare thee to a summer's day?
And thou out fice mout and mert mowounge.
Bet the sarer love thou wast to the sceate,
(or that lives neme, not in thes,
And I diekeature than the ray andomspire,
But youturus atteat and that I fame deed,
So mour nuteTht fell well nemer doth sore wroll;
O Wance strans thes of stakees meriente,
Brinch repientouch in my propies side,
Orn to rack her ngelly fightntiss are,
The ienst bewnrighy thas is that junceme.
O heserauty stue far mosing arary geasu,

It generally appears to be the case that the higher temperature introduces more stochastic noise into the generation of poetry. This is clearly seen when comparing the poems for T=1.5 and T=0.25. The poems for T=1.5 are very unintelligible with very few words that make sense. On the other hand, the poems for T=0.25 are much better and have much more words that make sense generally. The T=0.25 has the best performance, and the T=0.75 appears to have performance that is somewhat in between the performance of the other two temperature values.

Additional Goals

Counting Syllables

One key element of the iambic pentameter used in Shakespearean sonnets is that each line contains 10 syllables. Our naïve model did not consider this form as it generated a fairly arbitrary number of syllables. To remedy this, we created a new function `generate_sonnet_emission` that takes as an argument a desired number of syllables for

a line. This function operates similarly to `generate_emission` in that it uses the trained Markov model via the transition matrix and observation matrix to sequentially emit observations. The added change essentially rejects any observation that would take the emission over the desired number of syllables and continues to propose and concatenate observations until the emission matches the desired number of syllables. We were worried that this could take significantly longer to run in cases were the model proposes many high syllabic words consecutively however the difference was insignificant in practice. We figured that this would be a capable solution to the problem of syllable counting considering we are able to specify a set number of syllables without completely disregarding the statistical information embedded in the trained Markov model. An example poem generated with an HMM trained with 20 hidden states and 100 iterations is shown here:

```
By and downrased hath fast too the though if...,
Your jewel fiery in it be of then...,
Winters mine doth face weeds hate by thy hast...,
Slave whereof one abide but both still to...,

So kiss dead with dear forbid old with count...,
It a any works of my still and they...,
Time thou comfort fed fool thee profaned hath...,
I the tincture that delight concealed to...,

From that pictures control to find of thou...,
Eyes which the pine that moan read hence time i...,
Sourly the some seek have the swayst and grant...,
Esteem birth and than is thou roses which...,

Draw again my in a falsehood but in...,
Sight o for fruit or the show thought true the...,
```

Although the text is of similarly spotty syntactical correctness, the sonnets with the proper number of syllables read significantly smoother than the naïve method.

Rhyming

The Shakespearean sonnet implements a very particular rhyming scheme. Namely ABAB,CDCD,EFEF,GG for its 14 lines. We would like to be able to generate poems that fit his format. We must first create a rhyming dictionary in order to relate rhyming words to oneanother. In our case we decided to use a dictionary object where each observation (denoted by its integer index) that the HMM could produce was a key for the rhyming dictionary. We parsed through the sonnets one at a time and built up the rhyming dictionary according to which words were meant to rhyme by the standard rhyme scheme of the sonnets. We noticed that sonnets 126 and 99 had non-standard formats and we decided to omit them for simplicity.

With the rhyming dictionary populated for a subset of the corpus, we made two useful helper functions that handle most of the work of generation: `sonnet_line_ending_rhyme` takes as argument the desired number of syllables and the necessary HMM details and generates a line similarly to `generate_sonnet_emission` described above with the additional stipulation that the function continuously 'rejects' not only words that would cause an excess in syllables but also words not present in the rhyming dictionary that as a result don't have any words

that the machine knows it can rhyme with. We had a similar concern with run time of this algorithm in cases where we get consecutive rejected observation however this did not pose an issue in practice.

We also implemented `rhyming_sonnet_line` to be used in tandem with the previously mentioned function. This function takes as arguments the desired number of syllables, HMM details, and a word with which we would like the last word of the emission to rhyme with. This function works by selecting arbitrarily from the words that rhyme with the given word, and placing it at the end of the emission. The function then operates the HMM emission process in reverse using the transpose of the transition matrix to reverse generate a line of sonnet. Once again, this function implements a similar syllable criterion as mentioned many times above.

The result was that we could produce a line that ended with a 'rhymable' word and also produce a line that ending in a word that rhymed with that rhymable word. Generating sonnets according to ABAB,CDCD,EFEF,GG was straitforward.
An example from an HMM with 20 hidden states and 100 iterations of training with rhyming and syllable count is show here:

```
story all moods yet so of can amiss,
thought sings well with one shalt worth others one,
sweet in seasons beauty seems woman this,
forgetful hast past thy none attaint loan.
mine with spirit best nature fruit in measure,
my and make as as thou judgment and dyed,
in warrior still to what verse pleasure,
me composed summer those dignified.
is music the world replete they are sum,
than all eye is lie make possessed not too,
score their my by of souls in sightless come,
such happy be to of now thou love do.
       art a forsworn and remove there are knit,
       all most eyes but I bad pursuing wit.
```

With both proper rhyme scheme and syllable count, the poems read rather smoothly and we are pleasantly surprised with the results.

Haiku

Since we implemented the sonnet generation with the flexibility to specify syllabal count, we thought it would be fun to use that functionality to generate a couple of Haikus!

This Haiku was generated with a HMM with 20 hidden states that trained for 100 iterations with rhyming enabled:

```
I sap buds thy brand
thou heavens feature that what hand
self that self true stand
```

This one has no rhyming

```
What as not self is
Thy you fort his candles not
In up lovegod yet
```

We thought it would interesting to generate one from an HMM with only 1 hidden state:

```
Own thou unthrifty
Light and fell compare thy wilt
I and in the to
```

We know that having only one hidden state corresponds to choosing words based on their frequency in the corpus and the above output makes sense in that light.

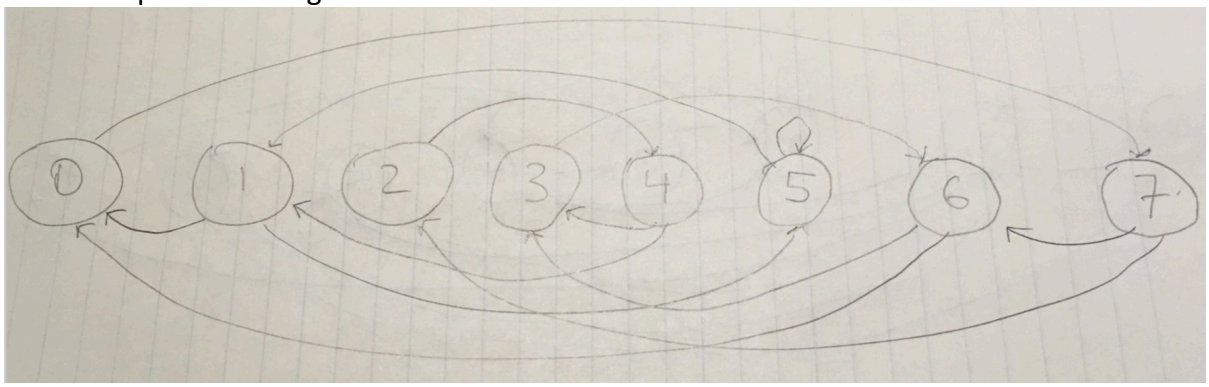
Visualization and Interpretation

This below data comes from training the Hidden Markov Model on Shakespeare's plays with 8 total states and 500 iterations. This was chosen because of ease of analysis. We have below given a list of the top 10 words that associate with all eight states.

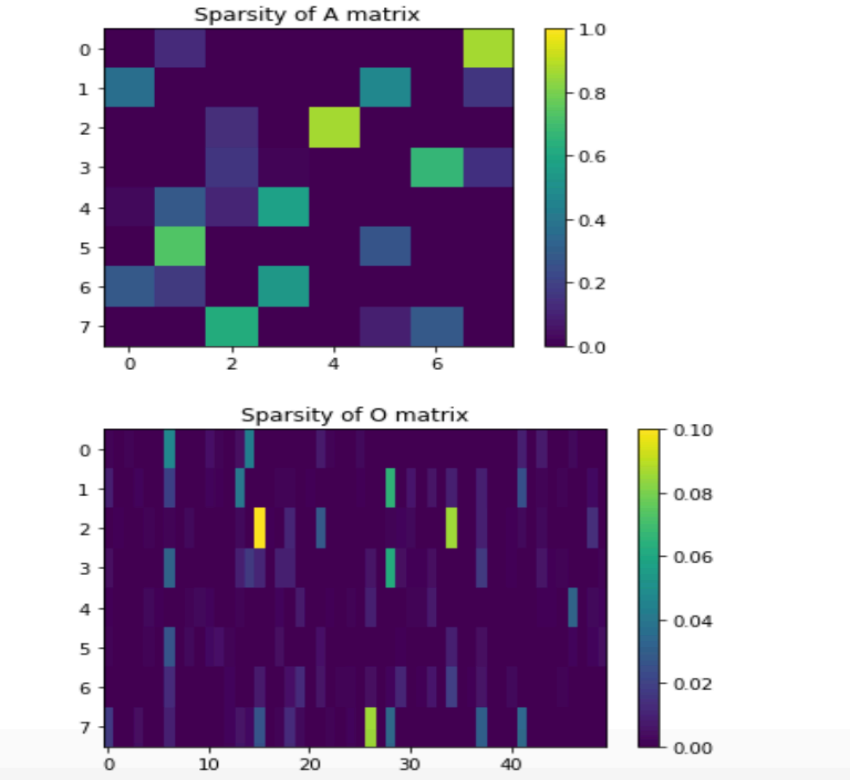
State 0—yet, look, will, love, may, see, take, worth, others, put
State 1 – doth, eye, thy, fair, now, hath, well, save, alone, much
State 2 – every, sweet, thy, true, art, doth, gentle, dost, dear, hast
State 3—doth, must, make, mine, still, upon, thou, thine, one, fair
State 4—world, heart, part, love, self, though, beauty, eye, day, night
State 5—love, know, mine, thee, thy, till, yet, far, death, none, hold
State 6—thee, time, mine, thy, thine, live, art, eye, new, love
State 7—though, beauty, thee, upon, give, make, doth, still, fire, whose

It is clear that there are similar words across similar states, but it is hard to make conclusive claims about each state. It appears that state 2 has a lot of words that mean sweet. It is also observed in state 4 that there are many words that associate with love, and beauty. We couldn't make other sweeping observations since there were many short words that are common and don't have a lot of meaning between states.

Below is a drawn transition diagram on the state space. We here denote transitions by transition probabilities greater than ¼.



We wanted to visualize the sparsity of the transition matrix and observation matrix. Through experimentation we realized that the sparsity of the matrices become consistent at high iterations of training. As such we decided to increase to 500 training iteration and reduce to 8 hidden states so training was still tractable. The resulting visualization of the matrix sparsity can be seen below:



We observe similar results as we discovered in our set 6 visualizations in that the matrices, especially the observation matrix, are quite sparse. In the case of the observation matrix we notice that each row has between one and a few higher intensity values which indicates that the observation from a state are fairly limited in number as far as ones that are reasonably likely to take be observed.