Machine Learning & Data Mining
Miniproject 3: Shakespearebot 5000

Caltech CS/CNS/EE 155
Released: March $5^{th}$, 2023

---

# 1 Overview

- In this project, you will attempt to generate Shakespearean sonnets by training a HMM on the entire corpus of Shakespeare's sonnets.
- Poem submissions for this miniproject are due 9pm on Monday, March $13^{th}$, via Piazza. The reports and code are due 9pm on Tuesday, March $14^{th}$, via Gradescope.
- Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem. Make sure to set sharing permissions to at least "Anyone with the link can view". **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.
- When accessing the Shakespeare text data from your Colab notebook, please do so via a link from the raw github data in the course github.
- You can work in groups of size 2-4. You may keep the same group as in miniproject 1 or 2.
- You may use any language you want, but you must submit a report including documented code that lays out everything you did. We recommend Python for this assignment.
- You are required to share one poem with the class on Piazza.

## 2 Colab Links

HMM Implementation
LSTM Implementation

## 3 Pre-processing (15 points)

For pre-processing, our group decided to go with simple CSV processing. We loaded the file from Git, went through every line of the file, split them into lists, removed all the spaces, removed the end-of-line characters, and removed the punctuation.

At this point, we had to make a strategic choice in processing our data to train our HMM on. Our first thought was to create a 2D array containing lists of poems with each list containing all the words of that poem. The other option was to create a single large "1D" array containing only one list with all the words from all the poems stored inside. We tried both strategies and realized that using a 2D array led to a faster run time as well as better outputs.

Analyzing the dataset, we noticed that the file was organized in poems with a number preceding each poem. We thought that this would be ideal to find a convenient way to split each poem.

To implement this split by poem, we went through every line, and if the line was of length 1 in terms of strings (therefore a number), we would replace it with a "NEWPOEM" string that we also included at the very beginning of the file. We then proceeded to 1-hot-encode the entire file using pd.factorize. This led to a mapping of "NEWPOEM" to 0 which enabled us to easily split the whole text into different lists (corresponding to a poem each) of words.

Working on generating Haikus, we had to use the syllable dictionary provided for us to count the correct number of syllables required. Unfortunately, we realized we needed some additional data processing. First, we noticed that we needed to lowercase all of our words, which already were except for "I". This does not appear on our output but is needed in the "lookup phase" when going in the dictionary. We then had to deal with some edgecases where apostrophes had been added to the beginnings or ends of words by shakesepeare, thereby leading them to not be in the dictionary. We solved this by removing apostrophes that were prefixes or suffixes of the word, and if they were in the dictionary we kept the words, and if not we reverted back to the original with apostrophes, which solved the issues that we had run into where poem words were not present in the syllable dictionary.

Finally, using the extra data set named spenser.txt required some extra data processing. Luckily, we were able to recycle most of the code used for the pre-processing of the Shakespeare file but had to find a new way to do the hot encoding. Since we wanted the hot encoding to be common to both the Shakespeare data set and the Spencer one, we had to group our words into one large array and then apply our pd.factorize() to the whole thing. This worked out well and we just had to fix some indexing issues that we encountered along the way. To test our data-processing steps, we simply used print statements and compared them to the original files.

# 4 Unsupervised Learning (20 points)

We did not use any packages from above and instead chose to continue with the code that we had written in HW6. Once we had preprocessed the data, with all the information one hot encoded, we trained the model. We started with 5 hidden states, and the proceeded to go up by multiples of 5 up to a total of 20 hidden states. The results of this can be seen in the colab link. As well, we attempted to train on a long single array that combined all the poems, however this saw an increase in training time with no visible benefit to the output and therefore we decided to continue with splitting the inputs into each poem separately as described above. This is likely do to fewer updates overall as $\alpha$ is only computed once, so more epochs could have made up for this decrease in performance but would have increased the training time relative to our implementation described above. As we increased the number of hidden states, the clarity of the poems increased and we therefore decided to use 20 hidden states as this had the best results.

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

## Naive Poem Generation from HMMs

Piazza Post

We used the unsupervised training model that we had developed in HW 6. This model involves using probabilistic models to predict an output for a system given an input. In our case, we mapped out set of words in Shakespeare's poems down to a number of hidden states (5, 10, 15, 20) in our case and attempted to learn two matrices A and O that would be used to predict words in generating our own Shakespearean poem. The A matrix provides probabilities for transitioning between from a given hidden state value to a new hidden state value. The O matrix gives the probability that given a "true" words from the given Shakespeare poems, the probability that word corresponds to one of the hidden states. These two probability matrices are then used in order to generate sequences of words, as outlined below.

Since we had removed punctuation in our preprocessing, we had to decide on a method for deciding when to create a new line. Since we had syllable information we decided that it was best to move to a new line whenever 10 syllables had been reached, in order to slightly preserve Shakespeare's practices for the sonnets. A 20 hidden states model generation is show below:

Away thou thy mistress with found but and
Spirit thou fair of at o this of their once
How is sweet love of wherein thought consent
Slavery yellow when sufficed read thou my
A with how dost mine sum stand know your their
Badges waking pyramids before chief
Countenance way homage none worth victors
Both foiled immured mine my graces doth
Might their these beauteous in in will it from
Shall the dull hue survive the think define

Numbers am of can if horse in unkind
Impute they which I question terms my might
Reeks can I commend every thy dear sin
I no of against the most look sweet have

The syllable count, as we enforced that rule, is obviously held in the sonnets that were generated. We see here that the rhyming scheme that Shakespeare used was not kept here. This is likely due to the fact that the generation of the words is in order and as it is a Markov chain it has no "memory" of the last words two lines above so would be very unlikely to see the rhyming scheme preserved for this type of generation. There are small sections of the poem that do make sense. It begins to lose clarity when the model encounters a preposition, after which it tends to string many prepositions together. As a result of this, we see that Shakespeares original voice is not preserved in the above rhymes. Without attempting to add line breaks whenever 10 syllables are hit and simply allowing the HMM to generate a line with a predefined number of words saw slightly less clarity while also having fewer groupings of prepositions. As a result we decided to keep working with 10 syllables per line for generation in order to keep some of Shakespeares original voice.

# 6 Additional Goals (10 points)

*"Though this be madness, yet there is method in't"* - Hamlet Act 2, scene 2

### LSTM EXPLANATION

To preprocess our data for the character-based LSTM we first started by removing all punctuation and newline characters/Converting all uppercase letters to lowercase from the Shakespeare text file while also converting all letters and spaces to values between 0 and 27, this was performed using stock-python operations. Then, to allow proper feeding into our LSTM it is necessary for all input sequences to be of like length, thus we divided our data into chunks of forty, treating the text like a time series where we slowly progress through characters in the text one character at a time, in arrays of forty characters at once, using basic numpy functionality. For our y-target we shifted our training set down a single character as we are attempting to predict the next character using our LSTM from the training data. Afterward, we constructed an LSTM with an Embedding, LSTM, and Dense layer. The Embedding layer is necessary as this 'layer' reduces our high-dimension forty character sequences down to a lower dimension, thus allowing our LSTM layer to properly accept our sequences as input. We used a hundred LSTM layers as this decreased training time the most. Finally, the last layer consisted of a Dense network that possesses a softmax activation function with twenty-seven units. We used 'Adam' as our optimizer with cross entropy as our loss function. We trained each model using four epochs as our collab-runtime lacked the necessary amount of RAM to withstand greater training length, our models were generated from the packages TensorFlow and Keras. Across our models we averaged around .5 accuracy validation accuracy wihtin our training. We then attempted to generate additional text after the phrase "shall i compare thee to a summer's day?", after using the same preprocessing techniques detailed earlier in this section we generated the following phrases using the listed

temperature on the soft-max activation function. We used functions and classes from both the Tensorflow and Keras module to generate our model.

| Temperature | Generation |
|---|---|
| Default | shall i compare thee to a summers day<br>derce then light tating doth all farifed<br>my ceen withor as kesecules tell it su |
| .25 | shall i compare thee to a summers day the<br>thertt iluve tpars evan tht the worchentt<br>waen thify puromt fire as when |
| .75 | shall i compare thee to a summers day yoy me priwy<br>dims nut telce out womerines doth<br>of love the sing to held pcuedes |
| 1.5 | shall i compare thee to a summers day seems<br>in that those thoush i carth detoy the wos<br>with likes you every with li |

As temperature increases, we start to see more coherent sentences with the model generating words from the dictionary instead of generating garbage text. In the case of the low-temperature model. Furthermore, the LSTM seems to generally understand the concept of 'words' in that there seems to be a consistent pattern with consonants and vowels. Otherwise, the model is unable to generate legible text in all of the generated sentences. It could be possible to increase the likelihood of text generated being from the dictionary if we increased the number of training epochs, but currently, since every group member is on the free-plan of google colab, we lack adequate processing power. We see that the poems generated by the LSTM poems are much more incomprehensible than those generated by the HMM model. However, if we attempt to decipher what the LSTM model was attempting to write, there is a glimmer of a comprehensive poem as it is attempting to build full words. This lack of clarity is to be expected as we are training over letters as opposed to training over full words so the model has to generate the inside of words as well as the transitions between words as opposed to the HMM which only has to focus on generating the next words in the sequence. Lastly, the training times of both the HMM and LSTM are both somewhat similar, with adequate results being shown after training both models for around 25 minutes.

# 7 Visualization & Interpretation (15 points)

| States | Words |
|---|---|
| 1 | 'love', 'self', 'not', 'thou', 'heart', 'but', 'love', 'why', 'and', 'will' |
| 2 | 'of', 'to', 'the', 'so', 'or', 'be', 'own', 'for', 'and', 'eye' |
| 3 | 'of', 'with', 'all', 'in', 'have', 'and', 'on', 'a', 'from', 'he' |
| 4 | 'and', 'I', 'which', 'that', 'so', 'the', 'when', 'thou', 'for', 'where' |
| 5 | 'and', 'by', 'o', 'to', 'that', 'then', 'with', 'nor', 'am', 'my' |
| 8 | 'that', 'thee', 'did', 'thee', 'more', 'thy', 'be', 'mine', 'might', 'to' |
| 13 | 'I', 'thou', 'it', 'you', 'but', 'not', 'me', 'or', 'love', 'her' |

We managed to extract the top 10 words that associate with each hidden state in our training. One example that we got is 'love', 'self', 'not', 'thou', 'heart', 'but', 'love', 'why', 'and', 'will' for the first hidden state when training our HMM with a total of 10 hidden states. Across the different hidden states, we notice that many propositions are repetitive. This makes sense as prepositions are in general predominant in the poems and would therefore tend to associate with the different hidden states. Some examples are "why", "and", "but"... We also notice the presence of the most common verbs used in Shakespearean poetry such as "love", "will"... In the example that we gave above, we also have the noun "heart" which is also very common across the poems. These patterns are also apparent in the other most commonly associated words seen above. There does not seem to be an association between the hidden states and particular parts of speech as the model progresses and since most of these are prepositions we are not seeing an association with stressed on unstressed words either. For this training, we can also look below at the sparsity graph for the $A$ matrix for the trained data:

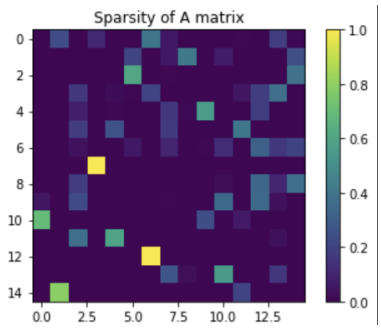


Figure 1: A matrix sparsity heatmap for 15 hidden states

Interestingly we see tha there is an almost 100% probability for transitioning from the $7 \rightarrow 3$ states (0 indexed, the above table is 1 indexed) as well as from $12 \rightarrow 6$. Given the content of these two transitions which can be seen in the above table, we have determined why our model had a tendency to go between prepositions whenever one was encountered.

The next section of this project deals with interpreting and visualizing the learned model. Our goal is to

interpret what the hidden states and transitions capture about the data. Use the learned observation matrix and transition matrix to determine what words associate most with each hidden state, and how the hidden states interact with each other. Do the hidden states represent parts of speech, stressed or unstressed words, number of syllables? What about anything else you can think of? You may use any open source tools to help you perform some of the analysis, such as NLTK.

In your report, you should explain your interpretation of how a Hidden Markov Model learns patterns in Shakespeare's texts. You should briefly elaborate on the methods you used to analyze the model. In addition, for at least 5 hidden states give a list of the top 10 words that associate with this hidden state and state any common features among these groups. Furthermore, try to interpret and visualize the learned transitions between states. A possible suggestion is to draw a transition diagram of your Markov model and give descriptive names to the states. Feel free to be creative with your visualizations, but remember that accurately representing data is still your primary objective. Your figures, tables, and diagrams should contribute to a discussion about your model.

# 8 Extra Credit (10 EC points)

Implementing up to two more additional goals (as described in section 7), you can earn up to 10 extra credit points. A maximum of five points will be awarded for each additional goal.

### Additional text incorporation

We decided to use a larger training set for our model to be more sophisticated (and hopefully accurate). We ended up adding the poems from the spenser.txt file to the poems from Shakespeare. Analyzing our results using our HMM from homework 6, we realized that for the same number of hidden states, the vocabulary was richer and not as repetitive in the use of pronouns. We were very satisfied for the result and decided to keep the extra training data in the final training of our HMM. Here is an example of a generated poem:

made on o'ercharged and is ne those I seem
writers cannot bright far rest but being pity lothsome
and to liberty thee change mind thou oft many
cruel trespass a her lodging of t'abide lovers
my await lead with I forsworn then do day
amearst a the love and false do a this
mercy is that my records to some be the
worth son thy her and be endless all lov'st
me unkind rest thou I angel burthens love on
we form there then and intent a the have
dost to powerful your like a love well
to thy mine doth shows ghost can hate well
dare my dark live all is pill desire most
the but an astronomy engrafted eternal power both not

## Haiku Generation

Since we already had a system for creating lines with 10 syllables, a logical next step was to create haikus from the training model. We used the same algorithm s above will slightly different parameters to generate these haikus, one of which is shown below:

why may fears to her
to they so and doth made straight
sleep his own knights true

We see that this specific generation is actually quite clear. Looking at some other generations we see that this type of model tended to put prepositions at the end of lines. This is to be expected as we regenerate words if the syllable counts go over the desired count for the line, so the model might have to scramble to put small syllable words at the end of lines to keep with the haiku parameters.