Machine Learning & Data Mining
Miniproject 3: Shakespearebot 5000

Caltech CS/CNS/EE 155
Released: March 1^{st}, 2024

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 11:59pm on Friday, March 8^{th}, via Piazza. The reports and code are due 11:59pm on Saturday, March 9^{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 one of the links provided on piazza or here: Shakespeare corpus, Spenser corpus, syllable dictionary (with explanation here).
- 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 Introduction

William Shakespeare is perhaps the most famous poet and playwright of all time. Shakespeare is known for works such as Hamlet and his 154 sonnets, of which the most famous begins:

Shall I compare thee to a summer's day?
Thou art more lovely and more temperate:

Shakespeare's poems are nice for generative modeling because they follow a specific format, known as the Shakespearean (or English) sonnet.[1] Each sonnet is 14 lines, spread into 3 quatrains (section with 4 lines) followed by a couplet (section with 2 lines). The third quatrain is known as the *volta*[2] and has a change in tone or content. Shakespearean sonnets have a particular rhyme scheme, which is *abab cdcd efef gg*.

Shakespearean sonnets also follow a specific meter called *iambic pentameter*[3]. All lines are exactly 10 syllables long, and have a pattern of unstressed stress. For example, the famous Sonnet 22 begins:

Stress	x	\	x	\	x	\	x	\	x	\
Syllable	Shall	I	com -	pare	thee	to	a	sum-	mer's	day?

Here, each x represents an unstressed syllable and every \represents a stressed syllable. Try saying it out loud!

The goal for this project is to generate poems that Shakespeare may have written by training a HMM on his 154 sonnets. His sonnets are available in the data file `shakespeare.txt`.

[1] https://en.wikipedia.org/wiki/Sonnet#English_.28Shakespearean.29_sonnet
[2] https://en.wikipedia.org/wiki/Volta_%28literature%29
[3] https://en.wikipedia.org/wiki/Iambic_pentameter

3 Pre-processing (15 points)

The first step is to pre-process the dataset before you train on it. How you pre-process is completely up to you. Here are a couple of questions to help you decide how to do pre-processing: How will you tokenize the data set? What will consist of a singular sequence, a poem, a stanza, or a line? Do you keep some words tokenized as bigrams? Do you split hyphenated words? How will you handle punctuation? It may be helpful to get syllable counts and syllable stress information from CMU's Pronouncing Dictionary available on NLTK. You might also find the file `Syllable_dictionary.txt`, provided in the data folder, to be helpful; please see the associated file called "syllable_dict_explanation" for an explanation.

Your report should contain a section dedicated to pre-processsing. Explain your choices, as well as why you chose these choices initially. What was your final pre-processing? How did you tokenize your words, and split up the data into separate sequences? What changed as you continued on your project? What did you try that didn't work? Also write about any analysis you did on the dataset to help you make these decisions.

4 Unsupervised Learning (20 points)

Now your task is to create a HMM for poem generation. **NOTE:** You do not have to implement this yourself.

Your best option is to use the Baum-Welch algorithm that you implemented in HW6 or from the HW6 solutions. You can alternatively use packages like hmmlearn (documentation here), which can be installed with **!pip install hmmlearn**, or pomegranate (documentation here), which can be installed with **!pip install pomegranate** (more installation instructions here). You can also use any other package you like but be warned that HMM packages for python tend to be poorly documented, so your safest bet is to use the HW6 solutions. When training, you should try training models with varying number of hidden states to see what works best.

Your report should also contain a section highlighting your HMM. What packages did you use, if any? How did you choose the number of hidden states?

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

Some theory

Remember that the core of a HMM is the transition matrix and the observation matrix. Given a current state y_0, we can generate the next state by randomly choosing a state from the appropriate row in the transition matrix based on the the probability of transitioning to that state. Now, with the next state, we can generate a word by choosing randomly based on each word's probability of being generated from that state.

Naive Poem Generation from HMMs

Write a program that generates a 14-line sonnet from your HMM model. You will need to choose one sonnet that you generate and share it with the rest of the class on Piazza under the tag `project3`. Note that the poem that you submit on Piazza does not need to be from naive poem generation, and can be from a later improved HMM model or a recurrent model (see the next section). However, the poem that you submit must be computer-generated. You may update your poems until the deadline for the project.

In your report, describe your algorithm for generating the 14-line sonnet. As an example, include at least one sonnet generated from your unsupervised trained HMM. You should comment on the quality of geneating poems in this naive manner. How accurate is the rhyme, rythym, and syllable count, compared to what a sonnet should be? Do your poems make any sense? Do they retain Shakespeare's original voice? How does training with different numbers of hidden states affect the poems generated (in a qualitative manner)? For the good qualities that you describe, also discuss how you think the HMM was able to capture these qualities.

6 Poetry Generation, Part 2: Recurrent Neural Networks (20 points)

Try doing poem generation with the same data using a recurrent neural network (RNN). A tutorial for RNNs can be found here. You can use PyTourch or any package you like. Although you are welcome to experiment with any recurrent model and language representation (see the following section), please follow these guidelines in this section:

- Train a **character-based LSTM** model. A single layer of 100-200 LSTM units should be sufficient. You should also have a standard fully-connected output layer with a softmax nonlinearity.
- Train your model to minimize categorical cross-entropy. Make sure that you train for a sufficient number of epochs so that your loss converges. You don't necessarily need to keep track of overfitting/keep a validation set.
- Your training data should consist of sequences of fixed length (40 characters is a good number for this task) drawn from the sonnet corpus. The densest way to do this is to take all possible subsequences of 40 consecutive characters from the dataset. To speed up training, using *semi-redundant* sequences (i.e. picking only sequences starting every n-th character) works just as well.
- To generate poems, draw softmax samples from your trained model. It may be interesting to play around with the *temperature* parameter, an RNN/LSTM hyperparameter which controls the variance of your sampled text. This is done by dividing the logits generated by the dense layers of the RNN/LSTM by the temperature before applying the softmax when sampling. More explicitly, if we have the logit vector $\mathbf{z} = (z_1, \ldots, z_i, \ldots, z_n)$, we transform it to we have the final probability of the ith element given by $\frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$ where T is the temperature (if you decide to play around with temperature, choose several values for $T < 1$ and $T > 1$).

Explain in detail what model you implemented and using what packages. What parameters did you tune? Comment on the poems that your model produced. Does the LSTM successfully learn sentence structure and/or sonnet structure? How does an LSTM compare in poem quality to the HMM? How does it compare in runtime/amount of training data needed to the HMM? Include 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", and comment on their differences.

7 Additional Goals (10 points)

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

The poems generated using the naive HMM are probably not very good as sonnets. In this section, you will explore methods of improving your poems or extending them.[4] **You only need to attempt one of the tasks below for full marks on this section**. If you have ideas for other improvements to the poetry generation not listed here, feel free to talk to a TA and work on it. The sky is the limit.

Alongside examples of poems generated, talk about the extra improvements you made to your poem generation algorithm. What problems were you trying to fix? How did you go about attempting to fix them? Why did you think that what you tried would work? Did your method succeed in making the sonnet more like a sonnet? If not, why do you think what you tried didn't work? What tradeoffs do you see in quality and creativity when you make these changes?

Rhyme

Introducing rhyme into your poems is not actually that difficult. Since the sonnet follows strict rhyming patterns, we can figure out what rhymes Shakespeare uses by looking at the last words of rhyming line pairs, and add this to some sort of rhyming dictionary. Then, we can generate two lines that rhyme by seeding the end of the line with words that rhyme, and then do HMM generation in the reverse direction.

Meter

One way to incorporate meter is by creating states that represent word stresses and limiting transitions between stressed and unstressed words. For example, if a word ends in a stressed syllable, its state should not transition to a state with words that start with a stressed syllable. You can also guarantee a syllable count by using supervised learning and labeling words by syllable and stress, and counting syllables when generating your poem. However, you may find that a more constrained HMM yields lower-quality sentence structure. If you use too many states, the HMM may lose variety in its generation. To find a happy medium, try semi-supervised learning.

Incorporating additional texts

A powerful feature of HMMs is the ability to combine texts from different sources, with potentially-silly results.[5]. We have also provided the Amoretti[6] by Spenser, a contemporary poet of Shakespeare. All 139 of Spenser's sonnets in the Amoretti follow the same rhyme scheme and meter as Shakespeare's sonnets.

Generating other poetic forms

It may be an endeavor to try to generate other poetic forms using your HMMs. Can you generate Haikus? How about Petrarchan sonnets, limmericks?

Improving recurrent models

There are many ways you can improve the basic LSTM model in section 6. For example, you can try using word-embeddings or other morphological representations instead of characters, as well as more complicated recurrent models. Evaluate your improvements over the baseline model both qualitatively and

[4] One approach is to hand-label some sonnets with specific states, thus making the resulting learning problem semi-supervised. Another approach is to try higher-order HMMs, such as 2nd order.

[5] King James bible mixed with SICP

[6] https://en.wikipedia.org/wiki/Amoretti

quantitatively (e.g. using the perplexity metric, here perplexity is a natural metric to use as it gives a quantitative description of how likely the next part of a particular sequence is to occur, which is similar to how these models work). You may use any publicly-available code and papers as long as you reference them appropriately.

Choose your own!

This project is meant for you to have fun and explore new ideas. Talk to the TAs about your own ideas of how to make the poems better, and try it out. We may award bonus points for creativity.

8 Visualization & Interpretation (15 points)

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.

9 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.

10 Additional Resources

- TED talk: Can a computer write poetry?
- Natural Language Processing Toolbox
- Unsupervised Rhyme Scheme Identification Hip Hop Lyrics Using Hidden Markov Models