Machine Learning & Data Mining
Miniproject 2

Caltech CS/CNS/EE 155
Released February 17th, 2017

Introduction

- **Group Members**
  Andrew Wang, Steven Brotz, and David LeBaron
- **Team Name**
  Double Boost

Tokenizing

The first step of the process was to figure out a good way to tokenize our data. To achieve this, we first split the input file, using whitespace, in order to compute the set of all words that appeared in the file. For our first trial, we did not remove punctuation or capitalization, so "word", "word?" and "Word", would all be tokenized as different. Using a dictionary, each unique word was stored and given an integer to represent it.

After learning our HMM on the sequences, we could then generate a sequence of integers with the unsupervised model, and use our dictionary to create lines in a poem. Initially, we treated each line as its own sequence, before trying to treat each poem as a sequence. We found that using lines made it easier to learn rhyme and meter, while using whole poems was better for thematic continuity. We finally decided to split on words based on poem so that our input consisted of list of lists where each list represents a single poem, and elements of each list correspond to words of that specific poem.

We also found that we had better results by stripping punctuation and capitalization from the words before tokenizing, as the previous method generated poems with caps and punctuation in incorrect places. The new method generated poems that made a bit more sense and flowed better.

Algorithm

We did not use any additional packages other than the provided homework 5 solutions. For syllable counts, we found two open-source functions that takes a given word and returns the number of syllables in that word.

There were several decisions that had to be made in optimizing the quality of the sonnet produced. We initially started with 50 states because we wanted a state for each combination of part of speech, either stressed or unstressed, and capitalized or not. However, upon analyzing the poems produced from these number of states, we found there to be some blatant grammatical errors (same words back to back, consecutive articles, consecutive prepositions, etc). One of the such generated sonnets using 50 states is as follows:

Greater bear my loving that fair love be
Thy blood which tell posting many is they
Hast but beauties uprear no whose as ye
And me can seeking survey way her pray

My of thy into most greedy which eye
Long against and sort whether of embrace do
Is those and and it rarities wail why
'now prove object three look it agreement to

That do when bastard joy and from astray
With showers rhymes by fearing all to me
Dully then the thou whose devise with they
Allurement being a sprite of where so thee

How but still to spoil cruel sequent alone
Slander's live but choose beauty sleep as one

We believe that due to the excessive number of states present, words that are used similarly in the English language were separated into different states which led to clashing of similar parts of speech and thus the grammatical errors (i.e. "and and", "the thou"). To compensate for this, we decided to strip the tokenized words of ending punctuation (apostrophes and dashes remained) and of capitalization (all words represented as all lowercase). Thus, we only had to have one state per part of speech and so we ended up deciding on 8 states corresponding to verb, pronoun, adjective, adverb, conjunction, preposition, noun, and determiner. After changing to a smaller number of states, our sonnets flowed better (example below).

Those more and embrace holy of death is
Agree from rough tongue modern seeming she
Like enmity think fire shape of this
Thus therein seem but confounds rich flit me

Fair love which in benefit tame grace win
All men can never be false and in sight
I mightst and endure thy love cruelty
Whence is this heart pilgrimage lest your might

Behold the gifts praise in my desire
My content you be presents that had prey
For fair me and prepare my admire
This pains me now for over due decay

Hath all angel's above and youth thy love
Hath the reason so I inquire dove

We also found that increasing number of iterations (we used 1000) bettered the coherence and quality of the generated sonnets. In addition, we chose to use both Shakespeare's sonnets and Spencer's sonnets to increase the amount of training data we could learn on.

Poetry Generation

We generated our poems line by line. Each generated emission corresponds to a single line of the sonnet. To mimic Shakespeare's 10 syllables per line scheme, we converted each generated emission containing numbers to the corresponding sentence and checked to see if that sentence had a total of 10 syllables. If so, that line would then be considered a valid line in our sonnet. We also incorporated rhyme into our sonnet to imitate the *abab cdcd efef gg* rhyming scheme of Shakespeare's (more on this in *Additional Goals*).

Similar to Shakespeare's sonnets, our final generated sonnet has a theme (love leading to pain and death). However, if we compare consecutive lines, we sometimes notice that there often seems to be a jump in logic between the lines. This may be due to the fact that in generating emissions sentence by sentence, we are not guaranteeing that the sentences tie together. If each emission were to generate entire poems as opposed to a single line, this issue may be not as pronounced. Moreover, if we look at lines of our poem individually, we may encounter some choppiness and unnatural wording relative to modern language. Much of this may be attributed to our HMM trying to learn Shakespeare's early modern language.

Visualization and Interpretation

State #	Most Probable Words (ordered left to right)									
1	as	is	when	thou	so	on	all	with	to	that
2	sweet	beauty	world	mind	eyes	be	is	when	self	heart
3	of	me	than	not	and	for	no	is	or	but
4	mine	with	me	my	a	his	it	her	their	your
5	in	she	a	look	be	he	you	thee	so	I

It is rather difficult to differentiate the states from one another. The general trend is that they all contain one syllable determiners such as "that" and "me" possibly because these words are the most commonly used words in the sonnets from our training data.

State 1 consists of mostly propositions (as, so, on, to, with). State 2 consists of more complex nouns (heart, world, eyes, mind, beauty). State 3 contains mostly conjunctions (than, and, or, but). State 4 consists of mostly possessives (mine, his, their, your). State 5 consists of mostly pronouns (she, he, you, thee, I). There seems to be some resemblance, albeit ever so slight, between the states and parts of speech as we envisioned in our decision to use only 8 states.

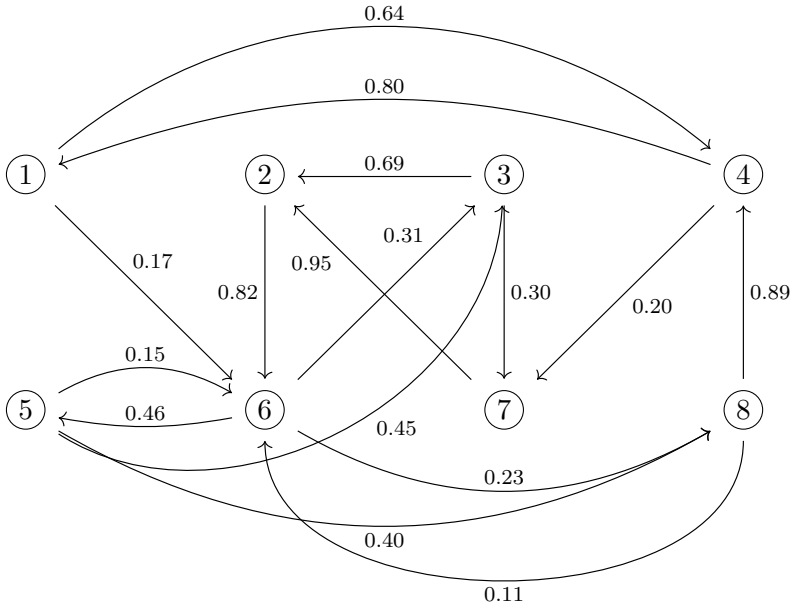

Figure 1: Only transitions with probability greater than 0.1 probability are depicted because any transitions with probability lower than that are extremely unlikely.

Removing all transitions except each state's most likely transition, we find

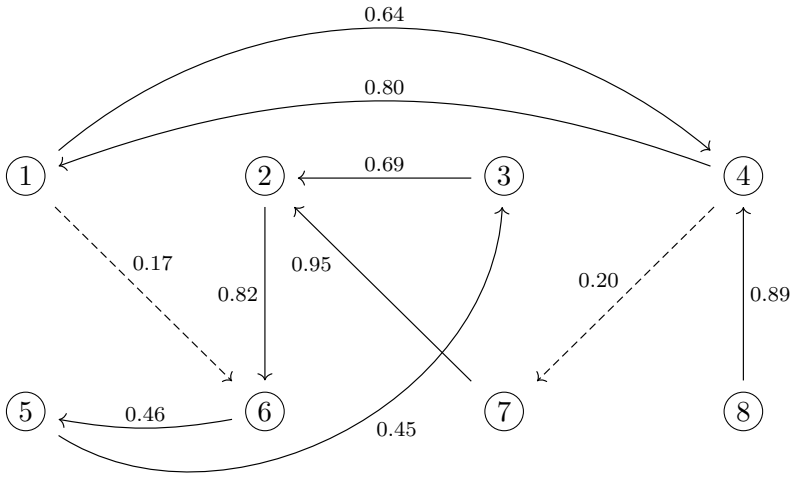

Figure 2: Only the most likely transition from each state is depicted, except for the dashed lines, which are the only transitions out of the 1-4-8 state group.

We may see that the transitions have probabilities that are reasonable given our analysis of which states they correspond to; conjunctions are followed by nouns with high probability, prepositions often precede possessives, etc.

Additional Goals

To incorporate rhyme, we had to incorporate a data structure that can keep track of rhyming words. First, we wanted to make sure that all poems were 14 lines long or we risk an offset in lines (that would invalidate our rhyming words). We wrote a script that computes the length of each sonnet in our dataset. We found that Shakespearean sonnet 99 and sonnet 126 were not 14 lines, and hence cast these sonnets from our training dataset.

Next, we went through each sonnet line by line. We took the last words of lines that were expected to rhyme (according to *abab cdcd efef gg*), made a tuple of these two words, and added them to a list that stores all pairs of words that are expected to rhyme. Now we iterate through this list of tuples and merge tuples that share common elements. We do this by converting each tuple to a set and for each set, we see if any other sets (pairs of words) intersect with it (meaning they share a common word). If so, we merge the two sets together by taking the union. This is our way of using the transitive property of rhyming in that if A rhymes with B and B rhymes with C, then A rhymes with C. Below are a few sets of rhyming words generated from the procedure described above:

[‘despise’, ‘miseries’, ‘lies’, ‘spies’, ‘subtleties’, ‘cruelties’, ‘devise’, ‘eyes’, ‘enemies’, ‘prophecies’, ‘arise’, ‘cries’]

[‘constancy’, ‘she’, ‘vain’, ‘posterity’, ‘thee’, ‘be’, ‘fine’, ‘melancholy’, ‘see’, ‘usury’, ‘agree’, ‘me’, ‘free’]

Note that there are cases where our rhyming sets contain words that do not rhyme with the rest such as ‘vain’ in the second example below. This is because although most of them do, not all of the Shakespearean sonnets follow the strict rhyming pattern. Because of such, we have ending words of lines that should rhyme according to the scheme that do not actually rhyme (‘love’ and ‘prove’ in Sonnet 10, ‘spread’ and ‘buried’ in Sonnet 25). This contaminates our rhyming sets a bit, but cannot be avoided based on the assumption that the rhyming scheme is perfectly accurate for all of our training data.

Getting the rhyming dictionary is the most difficult part of incorporating rhyme. Once we have this, we can generate emissions/lines that rhyme by testing if their ending words come from the same rhyming set. If so, then these are two valid rhyming lines in the poem and if not, we can regenerate lines until we get the rhyme we desire.

We played a bit with supervised learning and labeling words by stress, but found this to be overly tedious. In addition, we found it difficult to analyze the generated poems for correct meter.

Extra Credit

We did not make any meaningful attempt to complete the extra credit beyond doing some reading about recurrent neural networks (RNNs) and the long short-term memory architecture for RNNs.

Conclusion

David LeBaron planned out our approach for attacking this project (tokenizing, how emissions would be generated) and described such in the report. Andrew Wang did the majority of the coding and implementation and is the primary author of this report. Steven Brotz contributed heavily to the visualization aspect of HMMs by constructing the LaTeX diagrams, as well as making some small contributions to the code base.

We were pleased with the generated sonnets and realized that the tweaking of parameters (number of states, iterations, tokenizing method) actually does make a big difference in poem quality. Analyzing our generated sonnets proved to be a rather difficult task as Shakespeare's own writing is very hard to consider on their own. Although our generated sonnets may not have as good flow or command of language as Shakespeare's sonnets, they do a comparable job of capturing theme, syllable counts, and rhyme. Had we done more labeling and supervision, we could expect our sonnets to be of even better quality. Nevertheless, in doing this project, we now have a better grasp of the true extensive generative powers of Hidden Markov Models.