Shakespeare Chatbot

Jocelyn Griser

griser.jocelyn@gmail.com

March 12, 2023

Introduction and Problem Statement

Recurrent neural networks (RNNs) have been shown to be adept at text prediction for natural language processing. A well-trained model can take in information and then answer user queries given an understanding of language structure.

However, most of these models are trained on data with two important features: the training text is in modern language, and is structed with a singular "speaker" (i.e. the text is presented as if one person is providing all the information). The plays of William Shakespeare do not follow this format; Shakespeare's plays were written in the late sixteenth and early seventeenth centuries, and thus their use of the English language is distinctively different from modern English. They are also presented as (mostly) dialogues: two or more characters speaking back and forth. These differences may prove a challenge to established NLP model architectures as the data is not structured in the same fashion as the text the models are usually trained on.

The goal for this project was to create a "chatbot" that would utilize a model, trained on Shakespeare's plays, to answer a user's queries about said plays. The model would be fully constrained to the text of these plays, without influence of other pieces of knowledge or text.

Data

The data used for training the chatbot are the texts of eight of William Shakespeare's plays: The Tragedy of Antony and Cleopatra; A Midsummer Night's Dream; The Tragedy of Hamlet, Prince of Denmark; The Tragedy of Julius Caesar; The Tragedy of Macbeth; The Merchant of Venice; The Tragedy of Othello, the Moor of Venice; and The Tragedy of Romeo and Juliet. These eight plays were marked up into XML format by Jon Bosak and accessed through the NLTK Corpus library (`nltk.download('shakespeare')`). The XML format included play titles, speakers, personae, and the lines of dialogue. The dialogue lines were the data being observed and used, without distinction for speaker and without stage directions.

Research Design and Modeling Methods

Data

To begin, the size of each play was found, in both lines and words. In order to pare this to a more manageable size, the first 1,800 lines from each play were selected. These lines were then aggregated (for each play) regardless of speaker.

The aggregated lines were then cleaned by making all words lowercase and removing all non-alphabetic characters, including punctuation. These were then transformed into lemmas and had stop words removed. The stop words were taken from the NLTK Python Package (see Appendix “NLTK Stop Words”) and then the terms "shall","thy","thou","go", and "thee" were added to this set.

The aggregated lines were also then split into sentences using the `sent_tokenize()` function from the NLTK package. These sentences were also changed to all lowercase and had all non-alphabetic characters removed. (See Appendix Table A.1)

Exploratory

Exploratory data analysis was used before creating and training the chatbot model. TF-IDF analysis was run to find the “most important” terms across all documents. Most of the terms with the top TF-IDF scores (see Appendix; “Top 50 TF-IDF terms”) were generic verbs such as “come”, “make”, “let”, and “say”, and a few prominent characters: 'caesar', 'antony', 'cassio', 'romeo', 'brutus', and 'hamlet'.

Knowledge graphs were automatically generated using spaCy, a natural language processing library for Python, to analyze sentences to find “entity pairs” made up of a source and a target, and an “edge” that connected them. These relationships could then be plotted as a circular graph to show the links between entities. (See Appendix Figures A.1 and A.2 for examples.)

Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) were done to try and create coherent topics/concepts of a set number of words. These were created for 2, 4, and 8 topics/concepts, each with 10 words. However, the coherence for all of these was low (<30%) and the LDA concepts shared most terms with each other. (See Appendix Tables A.2 and A.3.)

Model Building

For building the model itself, the split sentences from `sent_tokenize()` would be used to create tokenized sequences. Using the TensorFlow Keras Tokenizer function, each sentence would be transformed into a sequence of integers, each integer representing a specific word in a dictionary created from the entire corpus. These sequences would be truncated to either 20 or 35 terms (depending on the model variation).

The sequences would then be split into n-grams: a sequence of n terms taken from a portion of the longer sequences. The n-grams were built to start with the first 2 terms in a sequence and then create a new n-gram, each adding 1 term, until the full sequence was completed. (See Table 1.)

Table 1: Sentence to Sequence to n-grams

Original Sentence:	`nay but this dotage of our generals overflows the measure`
Tokenized Sequence:	`[167, 19, 21, 2283, 5, 39, 1856, 4267, 1, 634]`

n-grams:	`[167, 19]`
	`[167, 19, 21]`
	`[167, 19, 21, 2283]`
	`[167, 19, 21, 2283, 5]`
	`[167, 19, 21, 2283, 5, 39]`
	`[167, 19, 21, 2283, 5, 39, 1856]`
	`[167, 19, 21, 2283, 5, 39, 1856, 4267]`
	`[167, 19, 21, 2283, 5, 39, 1856, 4267, 1]`
	`[167, 19, 21, 2283, 5, 39, 1856, 4267, 1, 634]`

In the model variations, the n-grams were restricted to either a maximum of 20 or a maximum of 35 terms, resulting in 86,290 or 93,334 total n-grams respectively. Due to limits on computational resources, out of these 30,000 random n-grams were selected for model training. (See Appendix Figures A.3, A.4, A.5, and A.6.)

For the models, four variations were created of a simple RNN. They consisted of an embedding layer, two (or three) Long-Short Term Memory (LSTM) layers, and two dense layers. Table 2 shows the formatting of each of the model variations. Each was trained for a maximum of 150 epochs but with an early-stopping mechanism if validation accuracy did not improve after 3 epochs.

Table 2: Model Architectures

Model	Model 1	Model 2	Model 3	Model 4
Longest N-gram	20 terms	35 terms	20 terms	20 terms
Total Tokens (TT)	8,987	9,574	8,987	8,987
Layers	Embed. 128 Bi-LSTM 128 Bi-LSTM 64 Dense, TT/2 Dense, TT	Embed. 128 Bi-LSTM 128 Bi-LSTM 64 Dense, TT/2 Dense, TT	Embed. 128 Bi-LSTM 128 Bi-LSTM 64 Bi-LSTM 32 Dense, TT/2 Dense, TT	Embed. 64 Bi-LSTM 64 Bi-LSTM 32 Dense, TT/2 Dense, TT

Chatbot

Once the models were fitted and evaluated, the model that performed best was selected to be used for the chatbot. The chatbot was hardcoded to understand certain formats of questions (ex. “Who are x?”, “Where was y?”) and transform those into “seed” texts (ex. “x are”, “y was in”). These seed texts would be fitted by the tokenizer and then the sequence was iteratively built onto by the bot by it taking the model and predicted the next term to the sequence, one at a time; i.e. the chatbot would take the original sequence and then predict the next token/word, and then would use the resulting sequence to predict the next token/word, and so on.

The chatbot could also ask if a person’s query was answered and would respond to the user text “goodbye” by ending the program. It could also recognize if a term in the user’s query was not in its corpus and ask the user to reword their query.

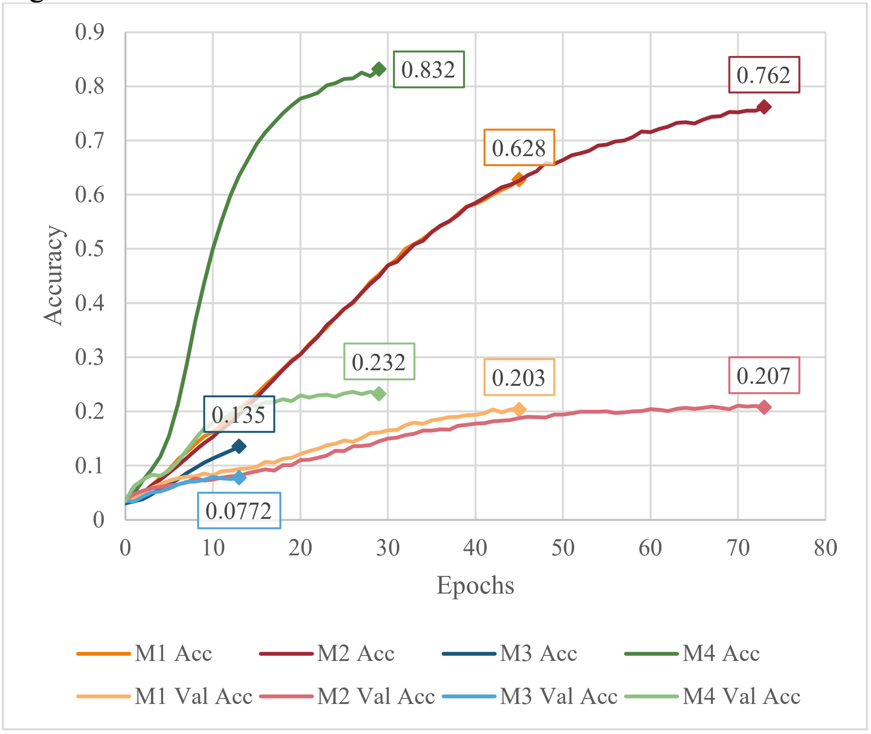
Results

Models

Table 3: Final Accuracy and Loss of Models

Model	Model 1	Model 2	Model 3	Model 4
Final Epoch	46	74	14	30
Acc.	0.6278	0.7620	0.1350	0.8315
Loss	1.9453	1.2950	4.8361	0.6189
Val. Acc.	0.2035	0.2071	0.0772	0.2324
Val. Loss	14.1597	15.8491	7.6558	14.9590

Figure 1: Model Variation Performance



Chatbot

Example Interaction:

```
SB: Greetings and salutations. I am ShakspeareBot.
SB: I will endeavor to answer your queries.
SB: To end our dialogue, say "Goodbye"

SB: What is your query?

Who is Juliet Capulet?

SB: "Juliet capulet is to the natural in the most
lamentable."

SB: Does this answer your query?

Who is Caesar?

SB: "Caesar is turnd knee many as which can hang."

SB: Does this answer your query?

No.

SB: I am sorry I was unable to answer your query.
SB: Perhaps try re-wording your query

Who was Caesar?

SB: "Caesar was turnd that in mexico known to make."

SB: Does this answer your query?

Who was Shakespeare?

SB: Forgive me. I am not familiar with the term
"shakespeare". Please reword your query.

Where is Rome?

SB: "Rome is in thee at the citadel and i do."

SB: Does this answer your query?

Tell me about Romeo

SB: "Romeo doff for you for ways cassio news."

SB: Does this answer your query?

Goodbye

SB: Farewell.
```

Analysis and Interpretation

Ultimately, the model(s) and the resulting chatbot were unsuccessful. The high accuracy but low validation accuracy as shown in Figure 1 mean that the models were extremely over-fitted; having a seed text that was an actual n-gram in the training set would be completely near-perfectly, but changing anything to a simple statement would lead to unintelligible sentences with improper grammar (for both Shakespearean and modern English) and mixed topics.

The models lacked any nuance from TF-IDF because of how they were built, and this can be exemplified by the query "`Who was Caesar?`" which returned "`Caesar was turnd that in mexico known to make.`" Mexico is mentioned only once in Shakespeare: in The Merchant of Venice. This play does not involve or even ever mention Caesar, so the model linking those two terms does not make logical sense.

The chatbot itself functioned as it should; it took user queries and built seed texts off of these queries, or responded by saying it did not understand a query or that it could not answer it. However, the model that did the actual text prediction, the reason for creating the chatbot, did not work at all as it should.

Conclusions

In order to create a successful and useful chatbot from this data, it would need to be trained on more data and/or had the data more carefully picked, such as by setting a longer minimum n-gram or by selecting more informational lines instead of lines at random. This would take more computational resources and time.

Using a straightforward RNN without TF-IDF may work some of the time, but due to the structuring of information in Shakespearean language it was difficult for the models to determine the key elements of the information. A more nuanced model that incorporated TF-IDF and other natural language processing tools could perform much more effectively.

Overall, ShakespeareBot was not successful, but did have the building blocks and potential to make an effective ChatBot. It is not impossible to use the tools of natural language processing to “understand” Shakespeare, but it would take a much more concentrated effort, including more time and resources.

Appendix

NLTK Stop Words: Stop Words from the English corpus of the nltk python package:

> [‘ourselves’, ‘hers’, ‘between’, ‘yourself’, ‘but’, ‘again’, ‘there’, ‘about’, ‘once’, ‘during’, ‘out’, ‘very’, ‘having’, ‘with’, ‘they’, ‘own’, ‘an’, ‘be’, ‘some’, ‘for’, ‘do’, ‘its’, ‘yours’, ‘such’, ‘into’, ‘of’, ‘most’, ‘itself’, ‘other’, ‘off’, ‘is’, ‘s’, ‘am’, ‘or’, ‘who’, ‘as’, ‘from’, ‘him’, ‘each’, ‘the’, ‘themselves’, ‘until’, ‘below’, ‘are’, ‘we’, ‘these’, ‘your’, ‘his’, ‘through’, ‘don’, ‘nor’, ‘me’, ‘were’, ‘her’, ‘more’, ‘himself’, ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’, ‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’, ‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’]

Table A.1: Lines, Word, Sentences by Play

Play	Lines	Words		Fewer Lines	Fewer Words	Sent-ences
The Tragedy of Antony and Cleopatra	3,560	23,573		1,800	11,989	1,452
A Midsummer Night's Dream	2,159	15,993		1,800	13,493	1,365
The Tragedy of Hamlet, Prince of Denmark	4,014	29,400		1,800	13,104	1,279
The Tragedy of Julius Caesar	2,596	19,057		1,800	13,516	1,413
The Tragedy of Macbeth	2,385	16,359	⇒	1,800	12,255	1,322
The Merchant of Venice	2,663	20,859		1,800	14,182	1,303
The Tragedy of Othello, the Moor of Venice	3,556	25,618		1,800	13,298	1,326
The Tragedy of Romeo and Juliet	3,093	23,798		1,800	13,823	1,495
TOTAL	**24,026**	**174,657**		**14,400**	**105,660**	**10,955**

Top 50 TF-IDF terms:

['come', 'good', 'well', 'love', 'make', 'man', 'lord', 'let', 'say', 'know', 'caesar', 'would', 'see', 'give', 'hath', 'ill', 'upon', 'speak', 'one', 'sir', 'must', 'yet', 'may', 'like', 'take', 'tis', 'antony', 'hear', 'think', 'night', 'time', 'tell', 'cassio', 'romeo', 'look', 'heart', 'eye', 'hand', 'much', 'brutus', 'death', 'none', 'mine', 'thing', 'friend', 'never', 'day', 'great', 'hamlet', 'leave']

Figure A.1: Knowledge Graph – Possessive (`edge= "'s"`)

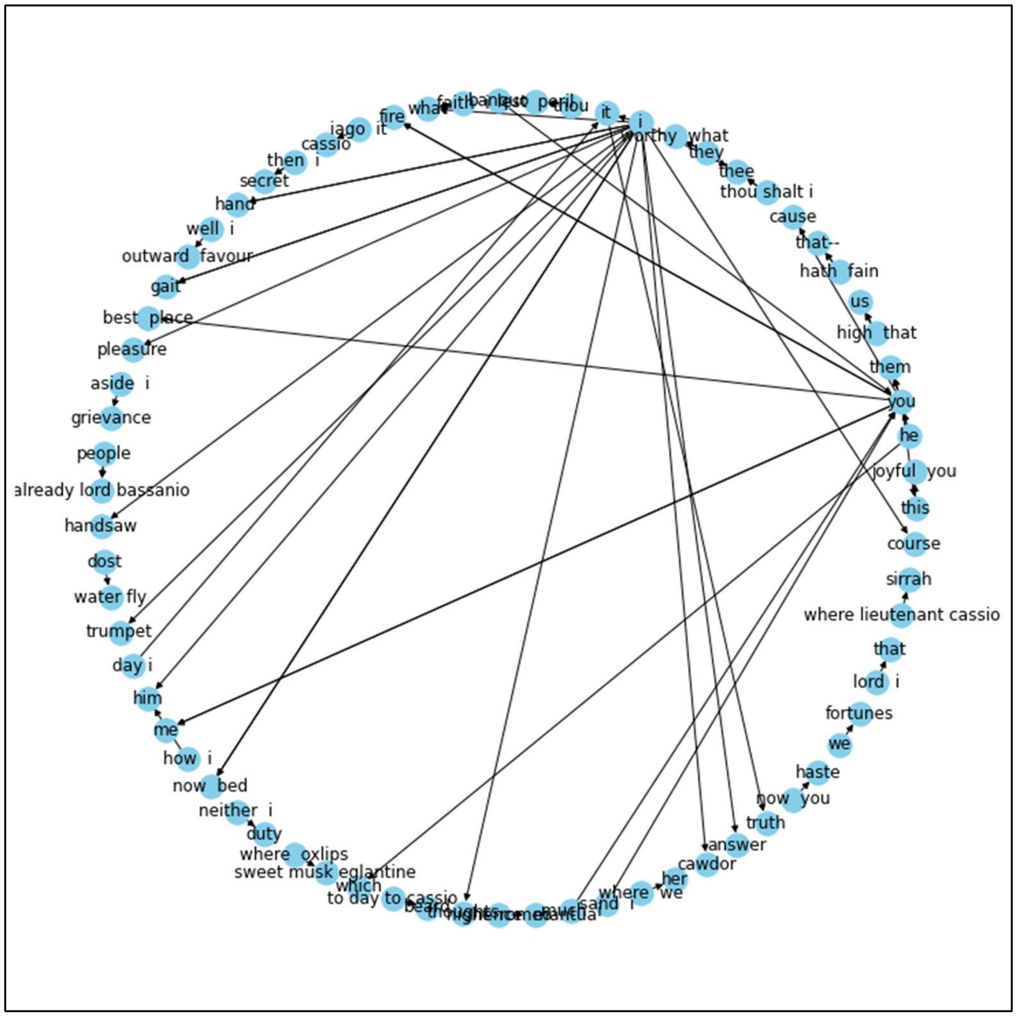


Figure A.2: Knowledge Graph – They (`source= "they"`)

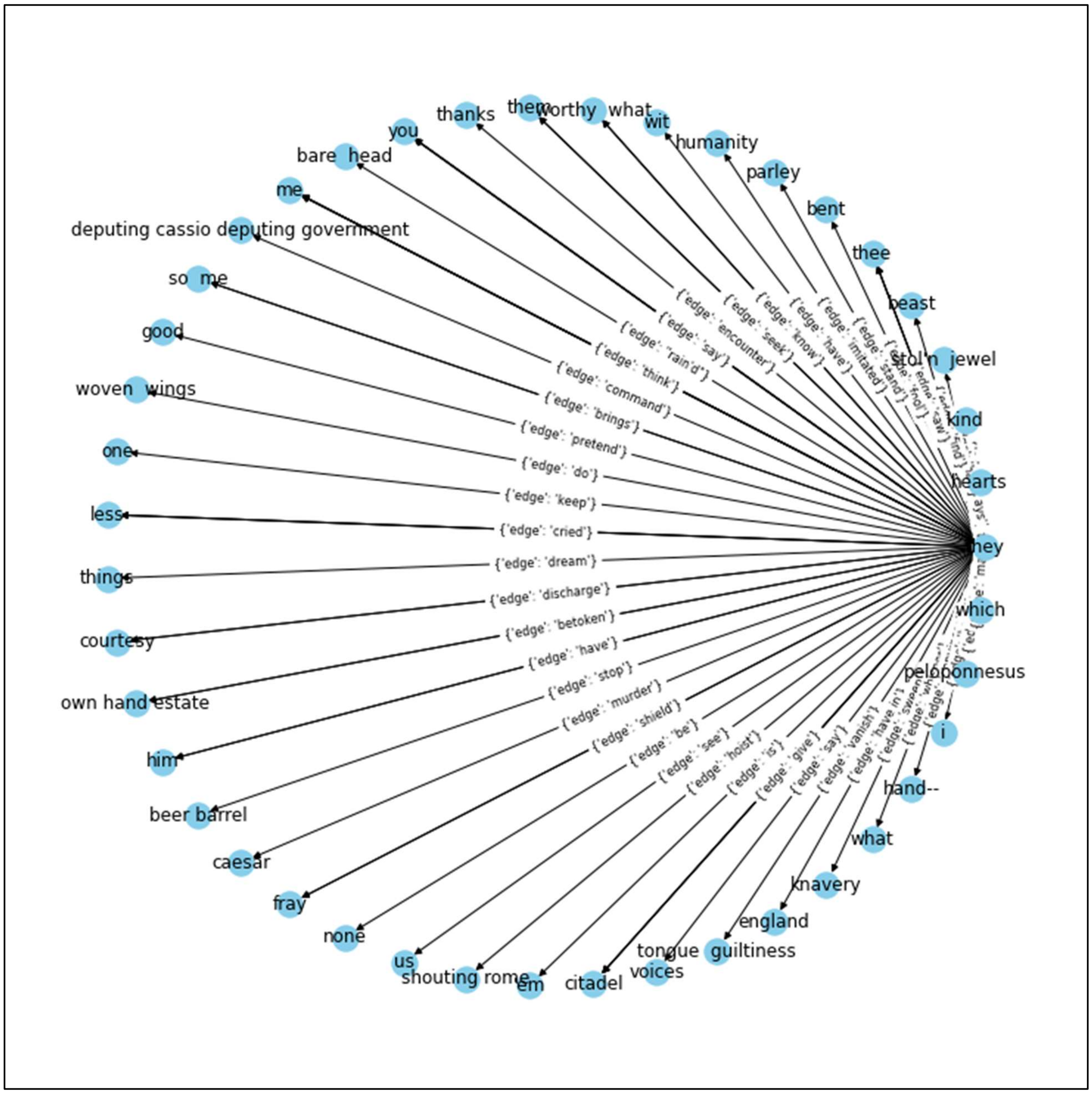


Table A.2: LSA Topics (Coherence)

Number of Topics (Coherence)	Topic Terms
2 (0.263)	0: 0.260*"come" + 0.211*"good" + 0.208*"well" + 0.196*"love" + 0.187*"make" + 0.177*"lord" + 0.174*"man" + 0.161*"let" + 0.155*"know" + 0.155*"say"'
	1: -0.606*"caesar" + -0.318*"antony" + -0.294*"brutus" + 0.187*"love" + -0.119*"rome" + 0.111*"lord" + -0.105*"cassius" + -0.105*"man" + 0.105*"romeo" + -0.097*"noble"'
4 (0.299)	0: 0.260*"come" + 0.211*"good" + 0.208*"well" + 0.196*"love" + 0.187*"make" + 0.177*"lord" + 0.174*"man" + 0.161*"let" + 0.155*"know" + 0.155*"say"'
	1: -0.606*"caesar" + -0.318*"antony" + -0.294*"brutus" + 0.187*"love" + -0.119*"rome" + 0.111*"lord" + -0.105*"cassius" + -0.105*"man" + 0.105*"romeo" + -0.097*"noble"'
	2: -0.384*"lord" + 0.278*"romeo" + 0.274*"love" + -0.165*"hamlet" + -0.155*"king" + 0.149*"man" + 0.136*"tybalt" + 0.132*"night" + -0.124*"tis" + 0.105*"death"'
	3: -0.401*"cassio" + 0.244*"lord" + -0.207*"iago" + -0.199*"moor" + 0.197*"hamlet" + 0.193*"king" + -0.141*"desdemona" + 0.138*"like" + -0.135*"think" + -0.121*"honest"'
8 (0.290)	0: 0.260*"come" + 0.211*"good" + 0.208*"well" + 0.196*"love" + 0.187*"make" + 0.177*"lord" + 0.174*"man" + 0.161*"let" + 0.155*"know" + 0.155*"say"'
	1: -0.606*"caesar" + -0.318*"antony" + -0.294*"brutus" + 0.187*"love" + -0.119*"rome" + 0.111*"lord" + -0.105*"cassius" + -0.105*"man" + 0.105*"romeo" + -0.097*"noble"'
	2: -0.384*"lord" + 0.278*"romeo" + 0.274*"love" + -0.165*"hamlet" + -0.155*"king" + 0.149*"man" + 0.136*"tybalt" + 0.132*"night" + -0.124*"tis" + 0.105*"death"'
	3: -0.401*"cassio" + 0.244*"lord" + -0.207*"iago" + -0.199*"moor" + 0.197*"hamlet" + 0.193*"king" + -0.141*"desdemona" + 0.138*"like" + -0.135*"think" + -0.121*"honest"'
	4: 0.340*"brutus" + -0.266*"antony" + 0.193*"man" + -0.166*"make" + -0.146*"madam" + 0.138*"lord" + -0.138*"sir" + -0.131*"romeo" + -0.129*"egypt" + 0.127*"cassius"'
	5: 0.223*"jew" + -0.205*"brutus" + 0.168*"antonio" + 0.153*"bond" + 0.152*"bassanio" + 0.151*"well" + -0.144*"romeo" + 0.138*"ring" + -0.128*"lord" + -0.124*"cassio"'
	6: 0.214*"romeo" + -0.197*"hermia" + -0.192*"demetrius" + -0.192*"pyramus" + -0.189*"love" + -0.178*"lysander" + -0.163*"eye" + -0.151*"play" + -0.136*"lion" + -0.135*"moon"'
	7: -0.246*"macbeth" + 0.197*"love" + -0.187*"fear" + -0.174*"thane" + 0.172*"lord" + -0.171*"yet" + -0.170*"upon" + -0.156*"banquo" + -0.150*"time" + 0.141*"caesar"'

Table A.3: LDA Concepts (Coherence)

Number of Concepts (Coherence)	Concept Terms
2 (0.246)	0: 0.010*"come" + 0.008*"good" + 0.008*"well" + 0.008*"man" + 0.008*"love" + 0.007*"make" + 0.007*"caesar" + 0.006*"let" + 0.006*"know" + 0.006*"see" 1: 0.010*"come" + 0.008*"well" + 0.008*"good" + 0.008*"love" + 0.007*"make" + 0.007*"lord" + 0.007*"man" + 0.006*"say" + 0.006*"let" + 0.006*"know"
4 (0.246)	0: 0.008*"come" + 0.006*"good" + 0.006*"well" + 0.006*"make" + 0.006*"man" + 0.005*"love" + 0.005*"let" + 0.005*"know" + 0.005*"say" + 0.005*"lord" 1: 0.010*"come" + 0.009*"love" + 0.008*"good" + 0.008*"well" + 0.007*"lord" + 0.007*"make" + 0.006*"man" + 0.006*"say" + 0.006*"let" + 0.006*"would" 2: 0.008*"come" + 0.006*"good" + 0.006*"well" + 0.006*"make" + 0.006*"man" + 0.005*"love" + 0.005*"let" + 0.005*"know" + 0.005*"say" + 0.005*"lord" 3: 0.009*"come" + 0.008*"caesar" + 0.008*"good" + 0.008*"well" + 0.007*"man" + 0.007*"make" + 0.006*"let" + 0.006*"know" + 0.006*"say" + 0.005*"see"
8 (0.247)	0: 0.008*"come" + 0.006*"good" + 0.006*"well" + 0.006*"make" + 0.005*"man" + 0.005*"love" + 0.005*"let" + 0.005*"know" + 0.005*"lord" + 0.005*"say" 1: 0.011*"love" + 0.011*"come" + 0.008*"well" + 0.007*"man" + 0.007*"good" + 0.007*"make" + 0.006*"say" + 0.006*"would" + 0.005*"see" + 0.005*"let" 2: 0.008*"come" + 0.006*"good" + 0.006*"well" + 0.006*"make" + 0.005*"man" + 0.005*"love" + 0.005*"let" + 0.005*"know" + 0.005*"lord" + 0.005*"say" 3: 0.013*"caesar" + 0.009*"come" + 0.009*"good" + 0.009*"well" + 0.008*"man" + 0.008*"make" + 0.007*"antony" + 0.007*"let" + 0.007*"know" + 0.006*"say" 4: 0.009*"come" + 0.008*"good" + 0.008*"well" + 0.008*"love" + 0.007*"make" + 0.007*"know" + 0.006*"say" + 0.006*"lord" + 0.006*"cassio" + 0.006*"let" 5: 0.008*"come" + 0.006*"good" + 0.006*"well" + 0.006*"make" + 0.005*"man" + 0.005*"love" + 0.005*"let" + 0.005*"know" + 0.005*"lord" + 0.005*"say" 6: 0.009*"come" + 0.007*"good" + 0.007*"make" + 0.006*"well" + 0.006*"upon" + 0.006*"know" + 0.005*"yet" + 0.005*"man" + 0.005*"would" + 0.005*"hath" 7: 0.012*"lord" + 0.010*"come" + 0.008*"good" + 0.007*"well" + 0.007*"make" + 0.006*"let" + 0.006*"know" + 0.006*"love" + 0.006*"give" + 0.006*"see"

Figure A.3: n-gram Length, max 20

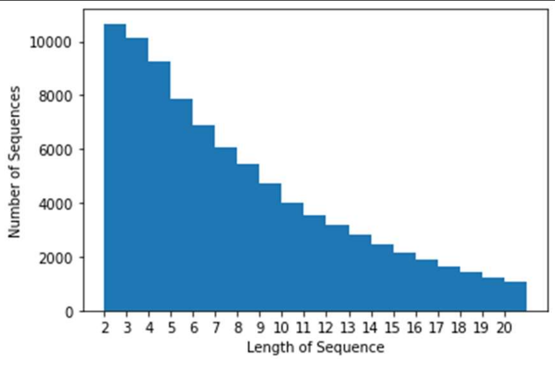


Figure A.4: n-gram Length, max 20, 30,000

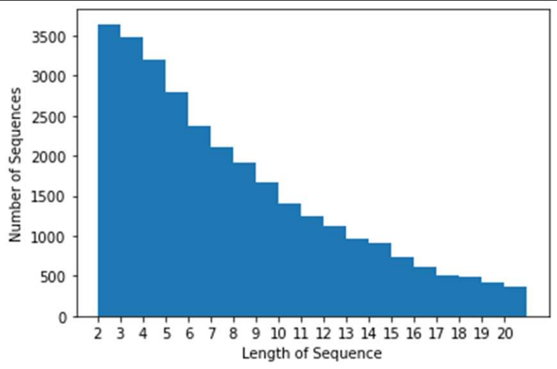


Figure A.4: n-gram Length, max 35

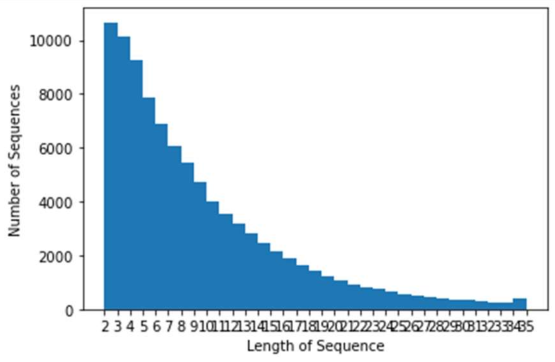
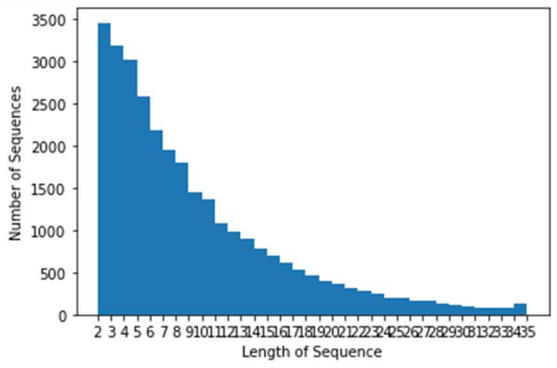


Figure A.4: n-gram Length, max 35, 30,000


Number of Sequences
Length of Sequence