AUSTRALIAN NATIONAL UNIVERSITY

Planning a Story

A Research Project On Computational Text Generation

Emily Rodrigo

5/26/2017

Acknowledgements

I would like to thank my supervisor Dr. Patrik Haslum for his help, advice, and support throughout the semester. I would also like to extend my thanks to the volunteers who participated in my experiments. Finally I would also like to thank Dr. Allen Rodrigo, for his help with the statistical analysis of my results.

Abstract

The aim of this project was to create a program that would take in a corpus of scripts and output a new script based on the structure and themes of the corpus. A secondary aim was to research and implement ways of increasing the coherency and sensibility of the generated scripts. The generator was implemented using Markov chains, and the Python topic modelling library Gensim was utilised in attempts increase sensibility. Experiments were performed by altering four different variables that affect various different aspects of the generator, and generating a data set of 25 scripts per combination of variables. These data sets were then tested for their length, similarity to the corpus, and sensibility. According to the sensibility test results there was no significant difference between the sensibility of the scripts generated using the methods of topic modelling and the control.

Contents

List of Figures

List of Tables

Introduction

On the 9th of June in 2016, a 9 minute science fiction film called Sunspring was released on the internet[1]. Sunspring was made for the Sci-Fi London film festival, and the unique concept was created by Oscar Sharpe and Ross Goodwin[i]. What makes Sunspring unlike any other film, is that it was written by an AI bot. The bot, named Benjamin, utilised a long short-term memory (LSTM) recurrent neural network (RNN) trained on science fiction movie scripts to generate the script for Sunspring.

The aim of this project was to 'recreate' the Sunspring experiment - to build a program that generates a brand new script based on a given corpus - with a secondary aim of improving the coherency of the generated script while still allowing it to be distinct from the source material. This report details the methods and algorithms used in this project, the experimental process and results, and the conclusions and future of this area of text generations.

Related Work:

Inspiration for the methods used in this project came from multiple areas of related work. `pt-voicebox` is a program created by Jamie Brew (GitHub user jbrew) that simulate the functionality of a predictive text interface one might find on a smartphone. It uses a "Markov-like" process to take in corpora files and output a list of 20 likely words to follow those chosen by the user. The results of this program can be found on Brew's Tumblr blog[2]. Other similar works include various text generators, ranging from simple Markov generators (`Markovify` by user jsvine[3], `markov-text` by user codebox[4], `Python-Markov` by user wieden-kennedy[5]) to more complex RNN based generators (`char-rnn` by Andrej Karpathy[6], `torch-rnn` by Justin Johnson[7]). Further related

[1] https://www.youtube.com/watch?v=LY7x2Ihqjmc
[2] http://objectdreams.tumblr.com/
[3] https://github.com/jsvine/markovify
[4] https://github.com/codebox/markov-text,
[5] https://github.com/wieden-kennedy/python-markov
[6] https://github.com/karpathy/char-rnn
[7] https://github.com/jcjohnson/torch-rnn

works can be found in the archives of NaNoGenMo[8] (National Novel Generation Month, a programming focused version of National Novel Writing Month), which feature various projects all based around text and story generation.

Algorithms

This project utilises Markov chains to generate text based on a corpus of raw transcripts. It also uses the Python library Gensim to create topic models that are used to assess the topics of the generated output and to make sure that the future output also has a similar topic model, implying that it is relevant to the lines that came before.

Markov Models:

Markov models are commonly used in text generation software. They provide a relatively simple way of generating text that is similarly structured but not identical to the source text they have been trained on. Markov models are much more accessible to those without experience in machine learning than neural networks, like the one used in the Sunspring project.

While they are good for text generation, Markov models can be used for many other applications. Their function at its most basic is simply to provide a model wherein the probability of the current state is conditioned on previous states in the chain[ii][iii]. For example, if a Markov chain is in state A, it may have a 20% chance of changing to state B, a 40% chance of changing to state C, and a 40% chance of staying in state A. This stochastic model is usually created by ‘training’ the model on a data set. So, in this example, the data set would more frequently move from state A to state C, than from state A to state B. This example would be a first-order Markov model. The basic formula for a Markov model can be described as follows:

[8] https://nanogenmo.github.io/

For any given list D of length c, comprising a sequence of states $S \in \{s_1, s_2, \ldots, s_n)$, a k-order Markov chain specifies the transition probabilities of seeing state $x(c+1)$ as $P(x(c+1)|x(c), x(c-1), \ldots, x(c-k+1))$.

In this project, the data set used is a text file, which the Markov module will then split into a list of words. The words in this list are then split into groups of three, with every word except for the first two and the last two appearing in three different groups of three. These triplets are again split into the first two words and the last word. The first two words become a key in a dictionary, while the last word becomes the value. If a pair of words occurs more than once in the text, then each word that occurs directly after it will be appended to the value list that corresponds to its key. This is a second order Markov chain.

When generating text based on this model, the generator first picks two words, a pair that will have a value in the dictionary. It chooses the following word as a random selection from the value list of pair, then it reassigns the values of those words. The first word becomes the second word and the second word becomes the randomly selected next word. This goes on in a loop until an ending condition - in the case of this program, a sentence ending punctuation mark - is reached. In this algorithm, the probabilities are determined by how often a word appears in a value list. Below is a simplified example of this process.

```
Data_set = [“I like cats”, “I like dogs”, “I like cats and
dogs”]
Markov_dict = {(‘I’, ‘like’):[‘cats’, ‘dogs’, ‘cats’], (‘like’,
‘cats’):[‘and’], (‘like’, ‘dogs’):[], (‘cats’, ‘and’):[‘dogs’]}
Output1 = “I like cats and dogs” (2/3 chance)
Output2 = “I like dogs” (1/3 chance)
```

Topic Modelling:

Topic modelling is a machine learning technique that is used for text-mining[iv]. The aim of a topic model is to produce a statistical model that presents the “topics” of a piece of text. The definition of “topic” in this context can be stated as “cluster of related words”. For example - “My dog was barking outside. I went out and saw that he had dug up a bone.” - might produce a topic that looks like: [“dog”, “barking”, “bone”]. There are many different methods of topic modelling, but this program uses the topic model centered Python library Gensim. Specifically, it uses gensim to create a Latent Dirichlet allocation (LDA) model. The LDA model classes the text into a collection of topics, each of which contains several words that are probabilistically related to the topic, along with the frequencies of those words[v].

The LDA model that can be created using Gensim has many different methods that can be used to view topics. The method used for this project was the `show_topics()` method, that outputs a list of topics, and the frequencies of the words associated with them.

The Script Generator:

The script generator is the main module of the program. It consists of a class called `Script`, which takes an open file as input. It creates an instance of another class called `TranscriptParser` using this file. The `TranscriptParser` class takes a file consisting of raw transcripts and splits it line by line into two separate files - ‘dialogue.txt’ and ‘stage_directions.txt’ - depending on whether the line contains a ‘:’ character (which implies that it is a dialogue line with the structure CHARACTER NAME: Line of dialogue…). The `TranscriptParser` also creates a list of single character strings, either an ‘l’ or an ‘s’, depending on whether the line it just placed was a dialogue line or stage direction line respectively. Finally, the `TranscriptParser` also creates a list of names taken from the dialogue lines. The `Script` class creates two separate Markov models using the Markov class and the two files output by the `TranscriptParser`.

The only method in the `Script` class is the `generate_script` method. When this method is called, first a 16 element sequence list will be created from the list output by the `TranscriptParser` using a simple third order Markov function. According to each item in that list, the script generator will either call the corresponding Markov model to generate a dialogue line or a line of stage direction. If the element is an ‘l’, a dialogue line will be generated, if it is an ‘s’, a line of stage direction. If the `Script` class uses one of the topic modelling approaches, the `generate_script` method will have a loop that will continue to generate lines until it finds a line that satisfies the conditions of the topic modelling function. This condition depends on how many of the generated lines LDA model chosen words match with those of either the previous line or of the entire priorly generated script.

The topic modelling function is used in one of two ways: either it generates a topic model of the generated line and the line directly previous, or it generates a topic model of the generated line and the entire script previous. It then creates a list of three topics, each containing three words, for both topic models, using the LDAs `show_topics` method. Each word in these lists is compared with each word in the other list to find how many words match. As stated above, there must be over a certain number of words for the function to return true.

If the function does not return true, the script generator will generate another line, and pass that through to the function. It can do this up to a certain number of times. If it does not find a line that passes the function in the given number to times, it will output the string “(END SCENE)”.

Finally, if the line to be generated is a line of dialogue, the script generator will choose a name from the `TranscriptParsers` list of names that has appeared previously in the script, and assign it to the dialogue line. If it cannot find a name that has appeared previously, it will choose a random name from that list, and assign it to the dialogue line.

Example output:

Below is some output from the script generator. They are examples of typical output for each of the different methods of topic modelling, as well as for the generator that does not use topic modelling. These scripts were generated based on a corpus of *Game of Thrones* transcripts.

```
DOLOROUS EDD. JON SNOW approaches MELISANDRE.

JON:
Now that you’ve brought to us, Jon Snow.

MELISANDRE looks at GREY WORM, and GREY WORM are sitting at the foot
of the Frey and Lannister men, listening. The men pick up JON SNOW’s
cheek. ALLISER THORNE are pointing crossbows at the hill.

JON:
You can stay here we’re dead men. Quickly, go, go!

Bronn turns and walks towards the Harpys, starts flapping and then
bows. INT. CASTLE BLACK - JON’S ROOM Jon goes through the
encampment, approaches a domed structure.

(END SCENE)
```

He begins taking her armor off. INT. LORD COMMANDER’S ROOM Jon and Grenn nod at each other. TORMUND runs towards Mago, while dodging his attacks.

While attacking, Mago screams. Khal drogo grabs Mago's throat, crushes it and stabs him through the snow on the ground, looks up at RAMSAY. RAMSAY notches an arrow and fires.

JON SNOW winces again. DOLOROUS EDD looks over at them.

JON:
Not again. Riverrun is ours. The scouts were wrong.

She looks up at her baby. RAMSEY whisles at the map with pawns Robb walks towards the gallows. DOLOROUS EDD approaches JON SNOW’s body remains on the steps behind him. As he drops to one side of the explosion.

The bell from the three guys attack Jon, trying to cut food. Sam looks to ROBIN. The surrounding knights begin to watch. Some people are spectating.

The militants, including Lancel, approach the main door to the front of the Faith Militant. CERSEI walks past their cage. Arya looks down the steps of the battlefield from one another.

RAMSAY:
How many wildlings got through- I already have one.

DAVOS sits on horseback wielding a whip, and one of the group move into a hallway, whistling. As he speaks, a CREATURE with glowing blue eyes.

WILL turns and sees a soldier laugh with each other, but before they pull their hands away. Nymeria slaps at Tyene and Nymeria are playing a game where one tries to stab RAMSAY with his party killed, also badly outnumbered. EXT. WINTERFELL Sansa lights a candle on a table.

Riding horseback and his men draw their swords JON and YGRITTE are hiding, waiting for Bronn. Bronn leaves Lollys and starts riding away as the door before Arya interrupts.

(END SCENE)

CUT:
Five Kings, they call it. A girl has taken the Seven Kingdoms, collaring pickpockets and horse thieves and runaways into men of the Vale!

He quickly turns around he sees she is to SEPTA UNELLA. MARGAERY slips a piece of meat. THOROS approaches and hands him one final time for Theon, while laughing.

MARGAERY:
Uncle Petyr! My lord.

At King's Landing in Ned's quarters. Arya and tosses out a scroll to Ser Meryn. Ser Meryn and Arya starts bleeding from the ground. TYRION turns around to face SANSA. MELISANDRE is sitting at one of his eyes.

TYRION:
You make me a doe, too, so that people will think she's pretty?

Silence occur for a second man. A volley of arrows. JON SNOW is pushes to the edge of the Wall.

Lord Commander Mormont leads them, but seems really bored by the shirt.

The Stark and Bolton armies sit across the arm and they both fall from their glasses. TYRION drinks all of his.

TYRION walks back and forth. BALON tries to flee.

JON:
Of course I want to bring me. (Picks up an X from the top of the Blackwater.

After a few pages. The maester puts down a hallway.

MARGAERY:
: I didn't think we'd find another way.

She looks at SAM and sits down. EXT. BRAAVOS - THE NEXT MORNING Stannis's men are arriving to the bed where Ned is not one of the tents.

TYRION:
Abomination!

INT. MELISANDRE'S TENT Melisandre stares into it, and a woman each is trying to choke him, throws him off, then goes inside.

Experiments

Experimental Aim:

The aim of this project was to create a text generator that generates scripts that are similar but not identical to the given source text, with a secondary aim of having these scripts make as much sense as possible. Thus, the aim of the experiments was to test several different variables and combinations of variables in order to find the combination that generates scripts that most match this criteria.

Variables:

There were four variables that were tested. The first variable was the method of topic modelling, of which there are two. The first method is to create a topic model of the generated line and of the line generated just before it and compare them. This method has been denoted the `Byline` method. The second method is to create a topic model for the generated line and for the entire script that preceded it, and compare them. This method has been denoted the `Scriptlist` method.

The topic models are compared in a function called compare, which iterates through the lists of words described in the Algorithms chapter, and checks how many of the words match. How many matches are required for the compare function to return true was another one of the variables. The values were 1, 2, and 3.

The script generator calls the compare function a given number of times in a loop. If it cannot find a suitable line using the compare function the given number of times, it will print the string "(END SCENE)". The third variable was this given number of times. This was tested in order to find out the minimum number of tries necessary to produce a script with five lines or greater.

The final variable was the order of the Markov model. The original Markov class was written as a second order model, but two more were created using third and fourth order models. The idea behind testing different Markov orders was to see whether it had a substantial effect on the grammatical correctness of the script without generating lines identical to those found in the source text.

The variables were selected so as to ensure easier and quicker script generation. For example, when the number of required matches is higher than three, it is more difficult to generate a line that will match, and thus the scripts are noticeably shorter. The same can be said for the number of attempts being lower than 25 - with less attempts it is harder to find a match. However, even at 75 attempts, the generator is noticeably slower.

Experimental Method:

For each different combination of variables (of which there were 54), 25 different scripts were generated. The scripts were generated in groups of five, with each group being generated from a different `Script` object. The 25 scripts were then saved in a file named to correspond to the variable combination. Also generated were three sets of 25 scripts using the plain script generator (i.e. without the use of topic modelling), with one set for each Markov order.

The observations performed on these generated scripts were done with the aim of determining which combinations generated the best scripts in terms of script length, where longer scripts imply that it is easier for the generator to find topic model matches, least similarity to the corpus, and greatest sensibility. The tests were as follows: taking the length of each generated script and averaging them over the dataset to find the combinations average length, using an eight word window to comb through the generated scripts and check how often the eight word strings appeared word for word in the corpus text, and finally giving sample scripts generated by the ten best combinations, as determined by the previous two tests, to five volunteers to rate the sensibility from 1 to 5.

Results:

Graphs of the average lengths for each combination based on the method of topic modelling (Figure 1) show a higher frequency of average lengths falling below 4 lines, but the `Scriptlist` graph clearly shows a greater range, implying that it is more likely to produce a longer script than the `Byline` method.

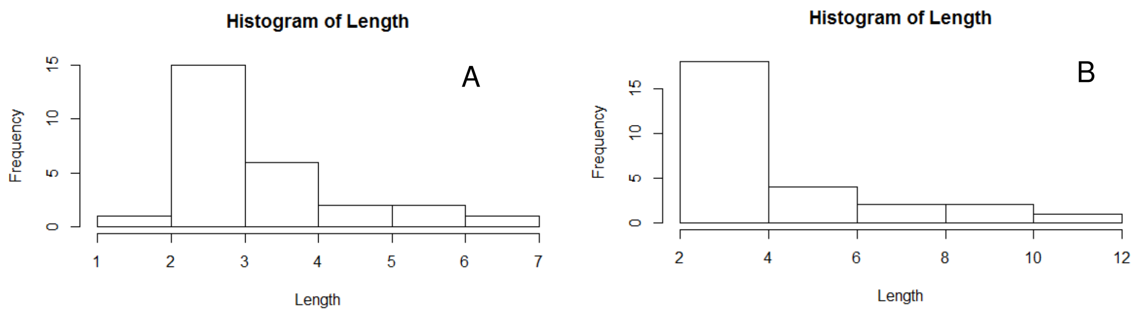
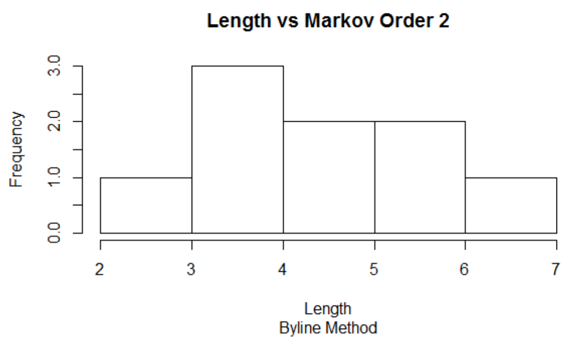
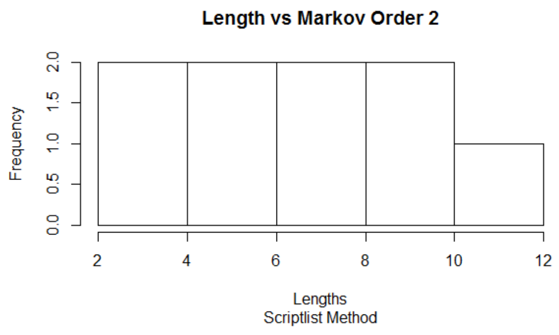


Figure 1. Histograms of average script lengths using (A) the `Byline` method and (B) the `Scriptlist` method.





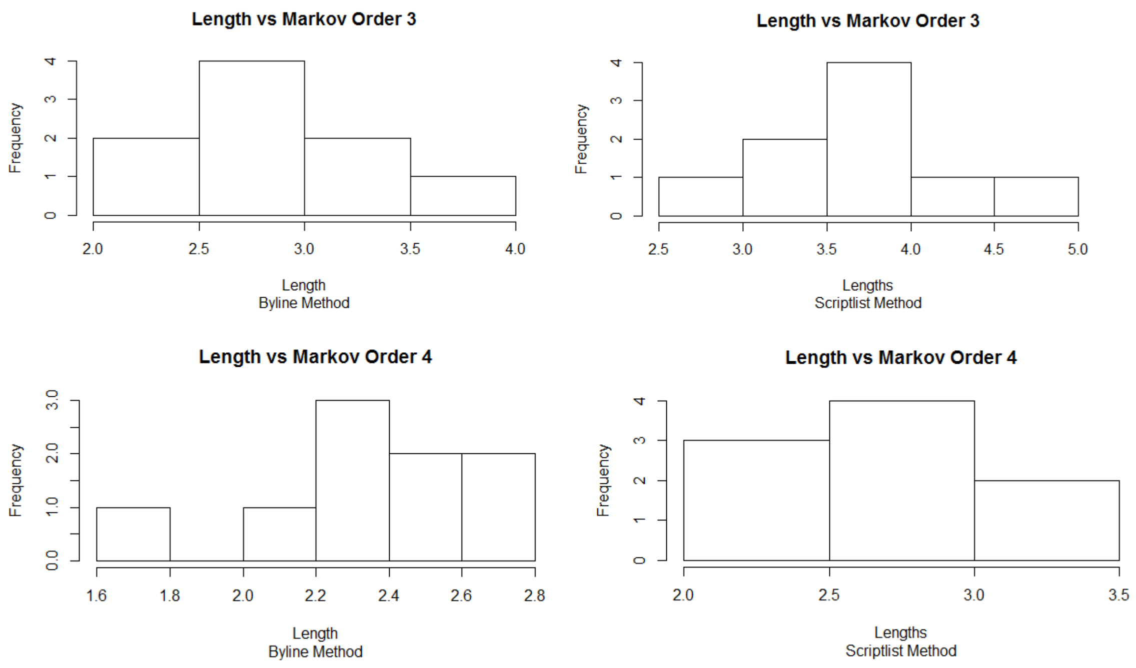
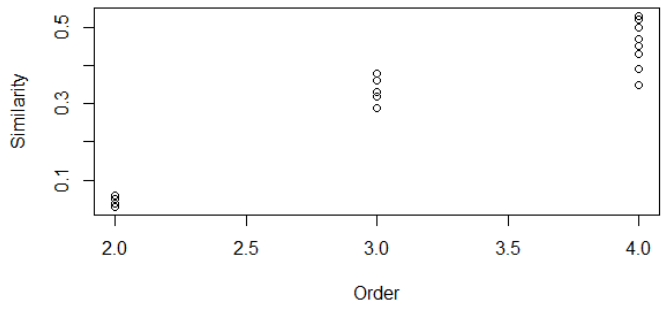


Figure 2. Histograms of average script lengths against Markov orders and topic modelling methods

Figure 2 displays graphs of the average lengths broken up further by Markov model orders. Higher orders such as 3 and 4 typically had lower averages, signifying the topic modelling conditions were more difficult to pass using lines generated by higher order Markov chains.

Byline Method



Scriptlist Method

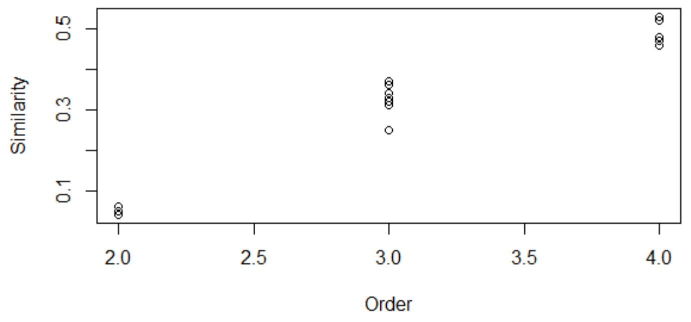


Figure 3. Plot of similarity to corpus against Markov order.

For both methods, combinations using higher order Markov chains had higher level of similarity to the corpus (Figure 3).

Below is the table of results from the sensibility test (Table 1). The first four columns represent the values of the variables used in each combination. To determine whether the use of topic models improved the sensibility scores when compared to a simple Markov model with no topic model (i.e., the control), I used a permutation test on the scores. The test was conducted in R, and by randomly permuting the scores of each model with those of the control. This was done 10,000 times, and the number of times the absolute difference between the observed mean scores was less the difference in the simulated mean scores was counted. If the proportion of times this occurs is less than 0.05, we can say that there is a statistically significant difference between the model and the control. After performing a permutation test with the plain generator against each combination, I determined that there was no significant difference between the sensibility ratings of the control and the combinations using topic modelling.

Table 1. Table displaying variable values and sensibility test result averages

Number of required matches	Markov order	Number of attempts at matching	Method	Average sensibility rating (range)
1	2	50	byline	1.9 (1-3)
1	2	75	byline	1.8 (1-3)
2	2	75	byline	2.8 (2-4)
1	2	25	scriptlist	2.8 (1-4)
1	2	50	scriptlist	2.3 (1-3)
1	2	75	scriptlist	1.9 (1-3)
2	2	50	scriptlist	2.0 (1-3)
2	2	75	scriptlist	2.6 (1-4)
3	2	50	scriptlist	2.2 (1-3)
3	2	75	scriptlist	2.4 (1-4)
N/A	2	N/A	plain	2.7 (1-4)

Discussion:

In this project, I investigated methods for generating scripts that were both unique and sensible. The scripts were generated using Markov chains, and the sensibility was regulated by methods of topic modelling.

Aside from the limits on the variables discussed above, the generated data sets were also affected by the transcripts themselves. Errors in the transcripts would translate to errors in the output scripts. In the file used for the experiments, stage directions that describe the location of the scene may begin with “INT.” or “EXT.”. The Markov module recognizes these as complete sentences, because they include a punctuation character. The algorithm also has an issue when regenerating a line if the line previous contains less words than the order of the Markov chain. This is because of the necessary reassigning of seed words when generating new lines that supposedly continue from the same key in the Markov chain. Finally, the results of the sensibility test were limited by the sample size of volunteers, which was five.

The results for the sensibility tests were largely inconclusive. While the other test results implied that the `Scriptlist` method and the second order Markov chain were better for generating longer and more unique scripts, there was no evidence that the topic modelling effort improved the sensibility of the scripts over the plain generator. This result is not unexpected however, as ‘sense’ is something very hard to define, especially so a computer can understand it. As showcased in the beginning of this paper, the generated Sunspring script, while generally grammatically sound, is still quite absurdist. This goes to show that there is still a long way to go in the field of text and story generation, if the ultimate goal is to generate text that is close to indistinguishable from text that may be written by a human.

However, there is an entertainment value in the absurdist and nonsensical output, as evidenced by the popularity of Sunspring, and things like the `pt-voicebox` blog. Generating nonsensical text can be a fun way to pass some time, or can even provide inspiration to get over a creative block.

Though it still requires more work, the benefits of sensible, structured text generation software could be quite far reaching, giving people the ability to generate not only scripts, but articles, manuscripts, advertisements, etc. It could also be of value to those developing automated chat bot or chat services.

Future Work:

To improve on sensibility, there is much more research into methods of topic modelling and uses of topic models that can be done. I would also be interested to see the results of the methods used in this project applied to a more advanced form of text generation, such as recurrent neural networks. To improve on the coherency of this script generator, I would like to do further work on text processing, so as to remove errors in the transcripts. Finally, research into grammatical structure in computational text generation would be beneficial, to improve grammatical sense.

Conclusion

The aim of this project was to create a script generator that produces unique scripts, and to research and apply ways to increase the coherency and sensibility of these scripts. The method used to generate the scripts was Markov chain based text generation. Topic modelling was utilized in attempts to increase sensibility. Experimental results showed that topic modelling methods made no significant difference to the sensibility of the scripts.

References

[i] https://arstechnica.com/the-multiverse/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/
[ii] http://setosa.io/ev/markov-chains/
[iii] Richard Serfozo (24 January 2009). *Basics of Applied Stochastic Processes*. Springer Science & Business Media.
[iv] Blei, David (April 2012). "Probabilistic Topic Models". *Communications of the ACM*.
[v] Blei, David M.; Ng, Andrew Y.; Jordan, Michael I (January 2003). Lafferty, John, ed. "Latent Dirichlet Allocation". *Journal of Machine Learning Research*.