**Project Title:** Movie Review Simulator 2016

**Final Project Report for CS 175, Winter 2016**

**List of Team Members:**
Arielle Chongco, 45137728, achongco@uci.edu
Anne Caballero, 21057614,alcaball@uci.edu
Carl Pacheco, 47911659, clpachec@uci.edu

## 1. Introduction and Problem Statement

The focus of our project is to create a random text generator able to emulate movie reviews. The main problem for our project is creating meaningful sets of texts from randomly generated words. Randomly generating words is easy enough. What is not easy however is making the generated text appear as if it were not. Some features of language that do not translate well with unprocessed text generation include: grammar, punctuation use, syntax, capitalization usage, and spacing. It is also difficult to make the generated text relate back to one central idea, goal, or meaning.

In the case of our project, this presents us with two main difficulties. Identifying the patterns in language that could help us create an algorithm which would help make our randomly generated text appear less so and connecting the generated review text to the movie it is supposed to be referencing and/or relating the review to a specific rating (positive or negative) depending on feasibility. For the former, instead of creating all the patterns ourselves, we left it to our Markov Generator to create patterns based on word frequency for one generator and part-of-speech frequency for another generator after a given sequence of words. For the latter we experimented passing review data specific to product and review data specific to a rating. We also experimented with the word sequence lengths the Markov Generators used to select the next set of words. From our project, we determined the best results word achieved with the following combination: basing the Markov Chaining off of words and not POS, using at least 3 long word sequences to base next word selection, and passing in data based on product rather than rating.

## 2. Related Work:

Random text generation is a problem that has been approached many times before and the methods typically revolved around Markov Chaining and weighing the occurrences of certain patterns in text. Sources that have explored these method include:

- A Random Text Model for the Generation of Statistical Language Invariatiants by Chris Biemann
    - http://www.aclweb.org/website/old_anthology/N/N07/N07-1014.pdf
    - In summary, the article discusses a text generator based on a distribution of several textual patterns instead of just generating words based on the frequency at which a word occurs in a given data set. Specifically the patterns Biemann mentions working with are: the distribution of word length, the distribution of sentence length, and significant neighbor based co-occurrence.

- Online Text Generators based on Markov Chaining:
    - http://projects.haykranen.nl/markov/demo/

    - https://golang.org/doc/codewalk/markov/
    - https://github.com/hrs/markov-sentence-generator

In relation to older work, for our project we applied the common method of Markov Chaining to generate text. In addition to that we also based the weights with which the Markov Chain algorithm picked the next word/POS to generate on the frequency the word/POS appeared after a given tuple length of words/POS.

## 3. Data Sets

As our project is to generate random movie review text, all our data sets are movie review datasets. However we were unable to retrieve live data as review APIs such as Rotten Tomatoes and Yelp only provide limited access to their live review data.

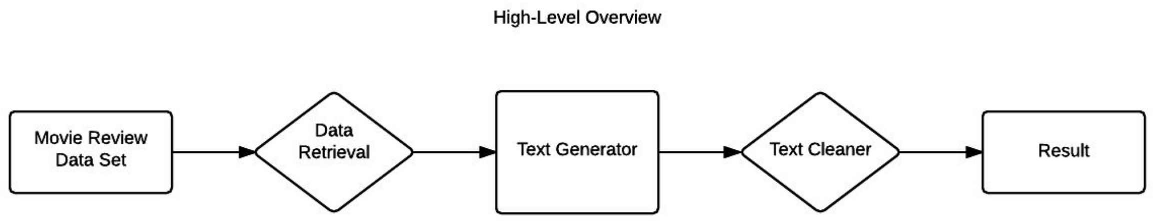
- Large Movie Review Data Set
    - http://ai.stanford.edu/~amaas//data/sentiment/
    - This is a large movie review dataset provided by Stanford consisting of 25,000 polar movie reviews for training and 25,000 for testing. It has been divided into two folders separating positive and negative reviews.
- Movie Review Data
    - https://www.cs.cornell.edu/people/pabo/movie-review-data/
    - This movie review dataset consists of 2000 processed down-cased text files, further divided into two folders dedicated for 1000 positive and 1000 negative processed reviews.
- Web data: Amazon Reviews
    - https://snap.stanford.edu/data/web-Amazon.html
    - This dataset consists of reviews from Amazon. Amazon provides Movies and TV reviews. It consists of 7,850,072 movie reviews. Unlike the other datasets, the data is not pre-parsed by rating. The dataset was obtained from reviews dated between June 1995 to March 2013.

| Large Movie Review Data Set | Movie Review Data | Web data: Amazon Reviews |
|---|---|---|
| **Excerpt from Negative folder:**<br><br>“I'm not a big fan of musicals, although this technically might not qualify as a musical. But I thought I would give it a chance as I love war movies. It was mediocre at best.<br /><br />Hudson seems totally out of kilter in this role. It just didn't work for | **Excerpt from Negative folder:**<br><br>“the happy bastard's quick movie review<br>damn that y2k bug .<br>it's got a head start in this movie starring jamie lee curtis and another baldwin brother ( william this time ) in a story regarding a crew of a tugboat that comes across a | **Excerpt:**<br><br>product/productId: B003AI2VGA<br>review/userId: A141HP4LYPWMSR<br>review/profileName: Brian E. Erland "Rainbow Sphinx"<br>review/helpfulness: 7/7<br>review/score: 3.0<br>review/time: 1182729600<br>review/summary: "There Is |

| | | |
|---|---|---|
| me. Julie Andrews probably played her part as best as she could, but..."<br><br>**Excerpt from Positive folder:**<br><br>"If you had asked me how the movie was throughout the film, I would have told you it was great! However, I left the theatre feeling unsatisfied. After thinking a little about it, I believe the problem was the pace of the ending. I feel that the majority of the movie moved kind of slow, and then the ending developed very fast. So, I would say the ending…" | deserted russian tech ship that has a strangeness to it when…"<br><br>**Excerpt from Positive folder:**<br><br>"after watching " rat race " last week , i noticed my cheeks were sore and realized that , when not laughing aloud , i had held a grin for virtually all of the film's 112 minutes . saturday night , i attended another sneak preview for the movie and damned if i didn't enjoy it as much the second time as the first . " rat race " is a great goofy…" | So Much Darkness Now ~ Come For The Miracle" review/text: Synopsis: On the daily trek from Juarez, Mexico to El Paso, Texas an ever increasing number of female workers are found raped and murdered in the surrounding desert. Investigative reporter Karina Danes (Minnie Driver) arrives from Los Angeles to pursue the story and angers both the local police and the factory owners who employee the undocumented aliens with her pointed questions…" |

Originally we used the datasets from Cornell and Stanford  to base our First and Second Generation Text Generators. However, for at least the Cornell dataset we noticed that all capitalization was removed from from the reviews and thus all text generation based on the dataset was lacking proper capitalization for pronouns such as names or locations. And both Cornell and Stanford did not retain the movie title which the reviews referred to which we needed to evaluate the difference between basing reviews on movie titles vs movie ratings. Thus for convenience we based successive generation generators on the Amazon dataset which had both a movie's rating and its title.

## 5. Description of Technical Approach



Through the course of our project we created four main generators with three following the pipeline pattern of retrieving data from the data sets, passing the data onto a text generator algorithm, and finally cleaning and formatting the string from the generator with the returned text being the final result. Our Generation 1 Generator was the only exception as it did not have any form of text formatting and cleaning as the most basic generator.

Data Retrieval:

For the Cornell and Stanford data sets, the data was already pre-parsed into ‘positive’ and ‘negative’ reviews. Thus when retrieving data from these datasets, our code did not have to account for finding rating and only required us to specify the directory location based on if we required ‘positive’ or ‘negative’ review data. We retrieved the data using python’s in-built file reading functions.

For the Amazon data set, the data was not pre-parsed but the data included review rating and title in a common format throughout the file. We parsed through the review data and stored it in separate file in dictionary format to more easily retrieve the data. For example, ratings were used as keys which could be used to retrieve reviews to be stored and passed to the generators.

Text Generation:

- Generation 1 (N-Gram Model): Placed data from Cornell and Stanford data set into table. After first word is picked out, each successive word is randomly picked based on the set of words the table has recorded as a possible follow-up word. For example consider a review data set where there were three strings which were “the apple”, “the dog”, and “the elephant”. If “the” was the first word picked out, there would be an equal chance that “apple”, “dog”, or “elephant” would be the next word generated.
- Generation 2 (N-Gram Model): Same as Generation 1 however add text cleaning with basic if-else statements to love for extra spacing to replace with individual space, properly capitalize words after sentence ending punctuations, and fix contraction separation due to using nltk tokenize() function.
- Markov Chain - Based on Rating: Placed data from Amazon data set into weighted tuple list. Data passed in is specific to rating selected which could be any number from 1 to 5. Tuples can be set to given length *n*. After the first *n* words are picked out to generate, each successive word is randomly picked based on how often it occurs after the given *n* words. The higher the frequency a word appears after the *n* amount of words, the greater the chance it has to be randomly generated.
- Markov Chain - Based on POS: Amazon data set passed through nltk’s POS tagger which is then placed into weighted tuple list. Data is also separated into dictionaries based on POS. Instead of chaining words, this version of the generator chained together POS tags which would then be replaced by words within the corresponding dictionary. A chain before word replacement would appear as: ['NNP', 'IN', 'DT', 'NN'].
- Markov Chain - Based on Product: Data retrieval and processing similar to Markov Chaining based on Rating, however the data passed in is specific to product number indicated.

Text Cleaner and Formatting:

For the Markov Chain based generators, the results were passed into a simple function which used regular expression to look for odd substrings which did not belong in reviews but were present in the datasets such as: <p>, </br, &quot&, etc. Double spaces were also removed.

## 6. Software

- (a) Our code
    - (i) Generator 2 Text Cleaning - Simple if-else script correcting generated string for spacing, capitalization, and contraction errors caused by tokenization.
        - (1) Input: Long string based on text from review data set
        - (2) Output: Randomly generated string of text with some corrections to capitalization, spacing, and contraction errors due to tokenization.
    - (ii) Parse Review (POS/Product + Rating)
        - (1) Retrieves data from Amazon review dataset
        - (2) Creates dictionary files. Product number and rating could be used as key values to retrieve corresponding reviews. POS could be used as a key value to retrieve corresponding word that fills in given POS key.
    - (iii) Markov Generator + Markov Word - Generate text based on chance of word appearing after given *n* amount of words. Higher frequency of word occurring after given word sequence increasing the chance of the word being randomly generated.
        - (1) Input: dictionary based on Amazon data set, Product/Rating, number of reviews to add to data pool
        - (2) Output: Randomly generated 'review' string
    - (iv) Process Text/Clean Text
        - (1) Input: string of text
        - (2) Output: formatted string with proper spacing and removed odd substrings that do not appear natural in generated reviews
- (b) Software and code from other sources
    - (i) Python 3.5. Scripting language
    - (ii) Eclipse, PyCharm, and Spyder IDEs
        - (1) Eclipse: http://www.pydev.org/
        - (2) PyCharm: https://www.jetbrains.com/pycharm/
        - (3) Spyder: https://pythonhosted.org/spyder/
    - (iii) N-Gram based Text Generation - Random text generation based on if a word occurred after another word in a given data set. No weights applied, therefore every word that is recorded to have occurred after a given word has an equal chance of getting picked for random generation. https://code.activestate.com/recipes/194364-the-markov-chain-algorithm/
        - (1) Input: Long string of review text
        - (2) Output: Randomly generated string based on given text
    - (iv) Natural Language Toolkit (nltk): Used for tokenization for Generation 1 and 2 Generators. Used for POS tagging for Markov Generator based on POS. http://www.nltk.org/

## 7. Experiments and Evaluation

Experiment Method: Process same or similar review data set with text generators to be compared against. The base generator is generator that was deemed the most successful from the last experiment and evaluation process with the exception of the very first generator. The generator to be compared against is the latest produced generator.

Evaluation Method: Evaluation based on user studies. Users were given results of base generator and the latest produced generator to compare results. The generator which was most prefered by users was selected to be the next 'base' generator for the next cycle of experimentation and evaluation.

Tables and results are available in the document uploaded alongside this one labeled "Generator vs Generator". A summary of the results we found that Markov Chain Generator - Based on Product (5th generation generator) provided the best results and was most prefered by users. From our results we also concluded with this generator, that basing generation on 2 to 3-tuple length would be ideal as with a smaller data set to base text generation on, 4 or more length tuples would greatly reduce the randomness of our text generation.

## 8. Discussion and Conclusion

Throughout this project we got to learn a lot about how text generation works and how much of text generation can be reduced to statistics. Particularly, we learned a lot about the Markov Chain and how it did not always have to be based on word frequencies but other features as well such as part-of-speech patterns and word lengths. We also learned more about nltk as we looking for different ways to approach our problem and discovered it had a convenient part-of-speech tagger which we used for one of our text generators.

The results were also somewhat surprising at the start. When we learned how to work with an N-Gram generator, we thought from there it would be easy to make a clearly "positive" or "negative" review if only we passed enough data into the generator even if there were no weights applied. However what we found was the results were wildly nonsensical. Similarly applying weights with only a 2-tuple based Markov Generator with weights appropriately applied to what text should be generated next after a given 2 word length sequence did not produce great results - although they were significantly better than no weights. It was only when we reached high order lengths of 3 or more that the generated text appeared more natural and related to its intended sentiment.

Besides that beginning though, most of our results were expected. For the most part, we thought the POS Markov Chain generator would likely fail against the word based Markov Chain generator as the words were separated entirely from the words they would normally occur next to. For example the word-based Markov Generator would likely know that “ice” and “cold” would be one pair and “hot” and “fire” would be another pair. The POS Markov Chain generator however would treat each word the same and would just as likely produce the pairs “ice cold” and “hot fire” as “ice fire” and “hot cold”. We also expected basing the data passed into the generators on product would produce somewhat more sensible reviews than basing the data on ratings.

Overall, based on the results of our project we concluded our code produced the best results with 2-3 order tuples and by basing the data set passed in on product being reviewed. However, the limitation is that it requires a large dataset which not even the Amazon dataset could entirely account for depending on how many reviews there were for a specific product. Reviews generated lose some ‘randomness’ as the data they have access to is limited. If we were to develop this project further, improvements we would likely work on would be finding a larger dataset and experimenting with Markov Chain based on word length. We would also research into other text pattern features that occurred in natural text.