Generating Startup Ideas with Markov Chains

Colton Gyulay cgyulay@college.harvard.edu
Antuan Tran antuantran@college.harvard.edu
Evan Gastman evangastman@college.harvard.edu

December 10, 2015

1 Introduction

Language modeling and text generation are important aspects of the greater field of natural language processing. In this paper, we use a naive language model based on Markovian assumptions to generate novel sentences from a text corpus. A major pain point for engineers and entrepreneurs lies in trying to determine what project or idea to work on. The specific application of text generation we examine in this paper seeks to alleviate this problem: our model creates fake, believable startup ideas using a dataset of company descriptions from the startup database Crunchbase. We've formalized the monkeys-on-typewriters approach to founding a company.

Our generative model is primarily based on Markov chains, which are decision processes that are characterized by the Markov property. A decision process is said to have the Markov property when any given state is independent of the preceding and following states. We can express this more formally:

$$P(X_n = x | X_0, \ldots, X_{n-1}) = P(X_n = x | X_{n-1}) \ \forall n \forall x$$

Though the traditional form of the Markov chain was first produced by mathematician Andrey Markov, the research our implementation follows is provided by Alexander Volfovsky, 2007.[6] When generating sentences, the biggest difficulties we had were ensuring syntactical correctness and ensuring a desired level of creativity (i.e., not creating sentences that in large part already appear in the training corpus). We were able to dramatically improve syntax by accounting for parts of speech during training and sentence generation. Part-of-speech tagging was done using a maximum entropy model trained on the University of Pennsylvania's Treebank corpus.[4] To ensure creativity, we used a bag-of-words model to prevent overlap beyond a certain threshhold between generated text and individual sentences in the corpus. The Model section provides a deeper discussion of these systems.

Choosing the correct state size (i.e., the number of tokens in a preceding state, k) was a critical part of producing strong results. A drawback of using Markov chains in language modeling is that they lack the ability to account for long-term dependencies in the text. For example, a subject and verb that are far apart—a greater distance that the state size, to be exact—may not agree in number. We found the ideal state size to be 3 or 4. When choosing a longer state size, these errors tend to disappear as generated text mirrors training text more closely. A shorter state size, however, confers the benefit of greater creativity. In the case of creating startup ideas, the more creative and whimsical, the better.

Using Markov chains for text modeling and generation fell in line nicely with CS 182's study of Markov decision processes and other probabilistic state transition models. Though deterministic in nature, our model was able to create novel and funny startup ideas—some of which we may be pitching to investors soon.

2 Related Work

The basics of Markov chains were explored through two primary sources: Alexander Volfovsky's paper on Markov chains and applications[6] and James Norris' book *Markov Chains*.[3] While Norris provided a more fundamental look at decision processes, focusing on the transition model and studying the underlying probability distribution, Volfovsky supplied a more accessible explanation in regard to application.

The most relevant work came from Grzegorz Szymanski and Zygmnut Ciota, who used hidden Markov models to build transition probabilities, and these HMMs to build Markov chains for generating text.[5] Their work also recommended some approaches for generating longer correct sentences by filtering difficult punctuation. Interestingly, Szymanski and Ciota employed a character level approach, which saw success with $k = 6$ or 7 letters. We found a word level approach to create the most consistent results.

Part-of-speech tagging was considered out of scope for this project, so we used the python library Natural Language Toolkit.[1] NLTK uses a maximum entropy model trained on the UPenn Treebank dataset, which includes standard tagged corpora like the Wall Street Journal and the Brown Corpus.[4] When augmenting our dataset, we also made use of the library Pattern for swapping verb conjugation between singular and plural. Pattern is based off of the work of Tom De Smedt and Walter Daelemans. [2]

3 Dataset

We should talk about our dataset here. This should include conversation on where we got it, how we preprocessed (removing sentences with bad punctuation like quotes or parens, pos tagging). Also we should talk about the data augmentation we did for multi-sentence generation (converting sentences of format companyname 3rdpersonverb to format We 1stpersonverb).

4 Model

Our model's primary algorithm constructs a dictionary of state transitions: it learns the underlying probability distribution by accumulating each preceding state's following state, and normalizing to 1. This process of accumulating preceding and following states essentially creates a phrase/word co-occurrence matrix M, where the phrase contains k words. See Algorithm 1.

The next important algorithm we implemented generates text given

Algorithm 1 Building transition probabilities.

```
procedure CONSTRUCT($sentences$)
    for each $sentence$ in $sentences$ do
        for $i$ in range(len($sentence$) - $k$) do
            $preceding \leftarrow$ tuple($sentence[i : i + k]$)
            $following \leftarrow sentence[i + k]$
            if $preceding$ not in $M$ then $M \leftarrow \{\}$
            end if
            if $following$ not in $M[preceding]$ then $M[preceding][following] \leftarrow 1$
            else$M[preceding][following] \leftarrow M[preceding][following] + 1$
            end if
        end for
    end for
end procedure
```

	Score
Approach 1	
Approach 2	

Table 1: Description of the results.

5 Body 2

6 Experiments

Analysis, evaluation, and critique of the algorithm and your implementation. Include a description of the testing data you used and a discussion of examples that illustrate major features of your system. Testing is a critical part of system construction, and the scope of your testing will be an important component in our evaluation. Discuss what you learned from the implementation.

6.1 Methods and Models

6.2 Results

For algorithm-comparison projects: a section reporting empirical comparison results preferably presented graphically.

6.3 Discussion

A Program Trace

Appendix 1 A trace of the program showing how it handles key examples or some other demonstration of the program in action.

B System Description

Appendix 2 A clear description of how to use your system and how to generate the output you discussed in the write-up and the example transcript in Appendix 1. N.B.: The teaching staff must be able to run your system.

C Group Makeup

Appendix 3 A list of each project participant and that participants contributions to the project. If the division of work varies significantly from the project proposal, provide a brief explanation. Your code should be clearly documented.

References

[1] Natural language processing with python. 2009.

[2] Pattern for python. *Journal of Machine Learning Research*, 13:2031–2035, 2012.

[3] James Norris. Markov chains. pages 1–56, 1998.

[4] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. 1996.

[5] Grzegorz Szymanski and Zygmnut Ciota. Hidden markov models suitable for text generation. 2014.

[6] Alexander Volfovsky. Markov chains and applications. 2007.