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.[7] 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.[5] 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[7] and James Norris' book *Markov Chains*.[4] 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.[6] 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.[5] 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]

We would be remiss not to mention a final inspiration for this project: legendary computer scientist and startup guru Paul Graham. His penchant for describing new ideas as the $X for Y$ (think "the Uber for laundry") as well as his essay "Ideas for Startups" make him the perfect example of a human-powered generative model of startup ideas.[3] We have simply extended his life's work programmatically.

3 Dataset

INCOMPLETE

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). Talk about how pos tagging disambiguates words with multiple use cases. Also we should talk about the data augmentation we did for multi-sentence generation (converting sentences of format "companyname 3rdpersonverb" to format "We 1stpersonverb"). We should also talk about padding sentences with start/stop tokens probably.

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. 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.

Algorithm 1 Builds 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 $preceding \notin M$ then $M \leftarrow \{\}$
            end if
            if $following \notin M[preceding]$ then $M[preceding][following] \leftarrow 1$
            else$M[preceding][following] \leftarrow M[preceding][following] + 1$
            end if
        end for
    end for
end procedure
```

The next important algorithm we implemented picks the following word given a preceding state by sampling from the distribution at that state. Each preceding state is represented as a row in M, where the total number of following states is equal to the number of times the preceding state appears in the corpus. By picking randomly from the total, the algorithm effectively normalizes to 1. See Algorithm 2.

Algorithm 2 Selects next word given previous state.

```
procedure NEXTWORD($preceding$)
    $total \leftarrow$ sum($M$.itervalues())
    $pick \leftarrow$ random.randint(0, $total - 1$)
    $accumulated \leftarrow 0$
    for each $key$, $weight$ in $M$.iteritems() do
        $accumulated \leftarrow accumulated + weight$
        if $pick < accumulated$ then return $key$
        end if
    end for
end procedure
```

The final key algorithm in our Markov chain model was the one used to generate sentences. It can generate sentences from an arbitrary starting state, though a blank start was primarily used to create sentences from scratch. The algorithm continues to accumulate words provided by Algorithm 2 until receiving a stop token. As a new word is added to the sentence, the preceding state is updated to include the k most recent tokens. This is where the startup ideas take their shape. See Algorithm 3.

After generating a sentence, creativity is ensured by setting a maximum threshold for shared vocabulary between the generated sentence and any single other sentence in the corpus. If there are too many overlapping words, the sentence is rejected immediately. In practice, a threshold

Algorithm 3 Generates startup ideas.

```
procedure CREATESENTENCE($preceding$)
    $traversing \leftarrow True$
    $words \leftarrow []$
    while $traversing$ do
        $following \leftarrow$ NextWord($preceding$)
        $traversing \leftarrow following \neq$ STOP
        if $traversing$ then $words \leftarrow words + following$
            $preceding \leftarrow preceding[1:] + following$
        end if
    end while
end procedure
```

	Score
Approach 1	
Approach 2	

Table 1: Description of the results.

ratio around 0.6 offered the right balance between creativity and semantic coherence.

5 Body 2

6 Experiments

INCOMPLETE

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.

LEAVING THIS TABLE IN HERE IN CASE WE NEED IT

6.1 Methods and Models

INCOMPLETE and might not be necessary given our Model section?

6.2 Results

INCOMPLETE and maybe unnecessary For algorithm-comparison projects: a section reporting empirical comparison results preferably presented graphically.

If we get some good multi-sentence stuff done we should def include.

Some favorite startup ideas include:

Anaphore develops protein therapeutics to treat diseases affecting dogs and breeding.

Focus Photography Inc specializes in state-of-the-art functional brain imaging that utilizes AJAX technologies in the development of Android apps.

The Australian soft toy for children and their patients.

Firalis is a fashion-based crowdfunding platform for gamers.

A123 Systems is an open community of genealogists collaborating to help users manage their bank accounts from one Web-based inbox.

Jewelry for the iPhone.

6.3 Discussion

INCOMPLETE

A Program Trace

Appendix 1 will demonstrate a simple example of building a co-occurrence table to model state transitions, and then generating a sentence using the observed probabilities. First, let's build the transition model. We'll use a token length of 1 and the example sentence:

The cat jumped on the cake and the cat ate the treat.

Given the word "the" as a starting state, the following states are comprised of "cat," which appears twice, "cake," which appears once, and "treat," which also appears once. The rest of the vocabulary is never reached from this state. The row of state transitions from "the" now looks like:

$$\begin{array}{c c} & \begin{array}{cccccccc} \textit{the} & \textit{cat} & \textit{jumped} & \textit{on} & \textit{cake} & \textit{and} & \textit{ate} & \textit{treat} \end{array} \\ \textit{the} & \left(\begin{array}{cccccccc} 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right) \end{array}$$

We can extrapolate the model to include the state transitions of the rest of the vocabulary:

$$\begin{array}{c c} & \begin{array}{cccccccc} \textit{the} & \textit{cat} & \textit{jumped} & \textit{on} & \textit{cake} & \textit{and} & \textit{ate} & \textit{treat} \end{array} \\ \begin{array}{c} \textit{the} \\ \textit{cat} \\ \textit{jumped} \\ \textit{on} \\ \textit{cake} \\ \textit{and} \\ \textit{ate} \\ \textit{treat} \end{array} & \left(\begin{array}{cccccccc} 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

For simplicity, the word "treat" has no available transitions, though in our model a stop token would indicate sentence completion. Now we can generate the first few words of a sentence. We'll again start with the state “the”. We’ll pick the next word following our observed distribution: “cat”: 0.5, “cake”: 0.25, “treat”: 0.25. Randomly we pick “cat” and our sentence is now “The cat”. We repeat the process with the updated state “cat”, which has the transition distribution “jumped”: 0.5, “ate”: 0.5. Randomly we pick “ate” and our sentence is now “The cat ate”. We can continue sampling words until a sentence end. This trivial example fails to demonstrate the creative capabilities of a Markovian language model, but with a significantly sized corpus each state will have many possible transitions.

B System Description

Appendix 2 explains the setup necessary to start generating startup ideas using our model. Start by cloning the repository here: `https://github.com/cgyulay/startup-generator`. Basic setup is outlined in the project’s `README`, but explanations will be provided here for completeness. An installation of python is assumed. The only python library required for sentence generation using a preprocessed corpus is `unidecode`. If using a non-preprocessed corpus, the libraries `nltk` and `pattern` are also necessary. To run part-of-speech tagging, it’s necessary to install `nltk`’s taggers from the command-line like so:

```
python -m nltk.downloader maxent_treebank_pos_tagger
python -m nltk.downloader averaged_perceptron_tagger
```

By default, the model will use a corpus of 50,000 cleaned and POS tagged sentences with $k = 3$. The corpus, token size, and number of ideas to generate can all be changed in `main.py`. To start generating the next killer startup ideas, run:

```
python main.py
```

C Group Makeup

INCOMPLETE 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] Paul Graham. Ideas for startups. 2005.

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

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

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

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