Generating Startup Ideas with Markov Chains

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

10 December 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.[8] 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.[6] 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[8] 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.[7] 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.[6] 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 $XforY$ (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

We collected 150,000 company descriptions from Crunchbase, all of them being one sentence in length. In cleaning up the descriptions for our use, we removed sentences with difficult punctuation, including question marks, exclamation points, quotations, parentheses, and brackets. As the Markov property is memoryless, if we have a sentence that contains a start quote, we are not guaranteed to have an end quote, which would make the sentence unnecessarily confusing and simply wrong. Similarly, we could possibly end a sentence with a question mark where it does not make sense with the earlier chosen words. Therefore, we removed the possibility of these confusions in the cleaning of the text.

Additionally, in the preprocessing phase, we used the python library Natural Language Toolkit (NLTK) to tag each of the words in the corpus with its part of speech (POS). This was helpful in generating more grammatically correct sentences because it resolved ambiguity where a single word could have multiple POS associations. Consider, for example, the word 'diet,' which can

be a noun or a verb. In our transition dictionary, instead of one key for 'diet', this word has two distinct keys, one as a noun and the other as a verb, such that when we request a next word according to the Markov model, the word is used in the right context with respect to POS. When we actually display this sentence after its been generated, all the POS tags have been stripped.

In order to generate multi-sentence company descriptions, we had to consider the language flow from one sentence to the next. As most of the company descriptions have the format "company-name 3rd-person-verb," it would make little sense if the second sentence began with a totally different company name, even if the description of that company was complementary to the first. To address this problem, we created a different model with the"company-name 3rd-person-verb," replaced with "We 1st-person-verb". This generalizes the descriptions since it removes the connections tied to specific company names in the word transitions. When generating a follow up sentence, we can use the generalized model.

Lastly, in creating complete sentences, we need to know where to begin and where to end. We used special start and stop tokens—unique character that do not appear anywhere in the corpus—to show where a sentence begins or ends. In making the transition dictionary, we append the appropriate number of start tokens to the beginnings of sentences corresponding to our token size and one stop token to the end of the sentences. This way, when we first choose how to begin a sentence, we have all of the first words as the values to which the start tokens can transition. As we generate the words, upon reaching a stop token, we are able to end the sentence and return it, knowing that we have reached a word that ends a sentence in the corpus. The creation of our transition dictionary from our data is covered in the next section.

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 ← sum(M.itervalues())
    pick ← random.randint(0, total − 1)
    accumulated ← 0
    for each key, weight in M.iteritems() do
        accumulated ← 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.

Algorithm 3 Generates startup ideas.

```
procedure CREATESENTENCE(preceding)
    traversing ← True
    words ← []
    while traversing do
        following ← NextWord(preceding)
        traversing ← following ≠ STOP
        if traversing then words ← words + following
            preceding ← preceding[1 :] + following
        end if
    end while
end procedure
```

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 ratio around 0.5 offered the right balance between creativity and semantic coherence. A deep dive into this process is outlined in Appendix A.

4.1 Multi-Sentence Generation

To create a contextually sensible follow up sentence to the first, we had a few strategies.

One initial strategy was to generate the second sentence with the same beginning word as the first. In almost all cases, the first word is the company name, so the first few words of the sentence would align, but once a different word entered the sentence, the two would diverge quickly.

In order to give the second sentence more freedom, we allow the sentence to begin anywhere but enforce an intersection of the two sentences in any non-common word in the sentence. The common words, which are a collection of words that we find are prevalent in the descriptions, are filtered out of one sentence then the two sentences are checked for a shared word. The second sentence is regenerated until such a match occurs. We find that these sentences are also not perfect complements to the initial, but they do provide a better-suited match since the matching word can influence the overlapping meaning in two directions —previous words and following words. This is superior to matching solely on the starts of the sentences, which tend to be more vague and can only influence the following words.

With the success of matching a word from one sentence to the next, we experimented with the next logical step; that is, matching multiple words. However, running this implementation and giving the second sentence even 100 tries to generate a sentence that matches two words yields results only 50% of the time for a training set with 50,000 descriptions. In our attempt to enforce more dependencies on the generated chain of words, we realize that we are trying to give a form of memory to a memoryless process.

5 Experiments

5.1 Single-Sentence Evaluation

When creating a brand new company description, we found the most success using token size $k = 3$. This produced the optimal balance of creativity and sensibility in our results. $k = 3$ enforces the grammatical correctness of phrases of at least four words, and our large dataset ensures flexibility when choosing the next word.

5.2 Single-Sentence Results

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.

> Good Health Media is an online platform that offers sports fans the opportunity to go on a fascinating wine adventure.

5.3 Multi-Sentence Evaluation & Results

Generating ideas with multiple sentences is often hit or miss. A example of a successful description where the two sentences align is:

> Iqua develops headsets and handsfree devices to enhance security, social media sharing, and advertising. We develop security devices and security systems in the industry today.

The idea in the first sentence is further developed in the second—the company makes the headset and devices with all these features, and their focus is shown to be the security feature. In this example, there are two overlapping words: ”devices” and ”security”.

Another success where the second sentence leads naturally from the first is:

> Nova Ratio develops and markets database automation software and solutions for digital signage networks. We develop software solutions for global organizations.

In this case, the specifics are not explained as much as the role and reach of the company. The first sentence describes what the company does and the second explains that its customers are organizations all over the globe.

A humorous case where the word matching is less than ideal is where the two words are ”social” and ”media”:

> Rentoid offers a rental site that lists apartments and flats, and a social media platform for pragmatic idealists working towards individual and collective progress. We are a social media website that rewards its audience with a virtual currency casino arcade that benefits game publishers, brands and media properties collect and analyze feedback.

There are four ideas in these two sentences, with two of them overlapping with the phrase ”social media.” The topic ranges from a rental site, to a social media platform for idealists, which transitions to a site with a virtual casino, and ends up helping companies collect feedback. The humor is that the social media site for idealists leads them to virtual gambling, and that the site is so multifaceted. With some creativity, one could connect the ideas into some sort of tangible proposal, but we would not consider the generated idea as successful as it far from a natural progression of thought.

5.4 Discussion

While we are excited about the progress we have made, the sentences we’ve generated, and the familiarity we now have with implementing Markov chains, we recognize considerable room for progress in areas like (1) improving our hit rate, and (2) enforcing semantic meaning in the generated sentences. First, we will address the success framework we used and then discuss ways to

improve semantic meaning going forward.

Sentence Success

We have not yet programmatically formalized what makes a sentence a success. This was due in part to the nature of the project and in part to the difficulty presented by possible evaluative techniques. We considered using crowd sourcing to select the best ideas, but we could not manage significant traction given our timeline. That said, we evaluated sentences based on their effect on us and our friends. Did a sentence make us laugh or cause us to scratch our heads and say 'hmmm', or did it simply not make sense? Again, though, this admittedly subjective evaluation is consistent with the nature of the project, which is meant to provide playful and perhaps inspirational content based on Markov chains. Successful sentences did have to pass one or more of the following tests:

- The sentence made us laugh based on the unusual blend of ideas from the sentences in the corpus.
- The sentence depicted a new company that could exist at present or in future.*ex. Soligenix develops biodefense vaccines and therapeutics for the prevention and treatment of breast tumors*
- The sentence managed to trigger a sequence of thoughts beyond the scope of the sentence. *ex. Datameer provides business users with an array of quilting supplies and a quilting channel on YouTube.* While this sentence struck us as comical because of the idea of a bunch of businessmen in suits quilting as they waited on the Datameer customer support line (passes the humor question), it also inspired us to think about potential new products that leverage Youtube as a teaching guide. A company that has a project-based learning channel on Youtube and provides content/materials for experiments in schools, was a non-trivial idea that came to mind. ...

Meanwhile, unsuccessful sentences failed primarily because they failed to preserve basic syntax such as grammatical number or failed to preserve context (placement of proper nouns in nonsensical places). At first, we thought an evaluative question would consider the 'boringness' of a sentence, but we quickly realized that sentences that struck one person as boring resonated with someone else. The nature of these sentences is that the reviews they get absolutely depend on the experiences and associations of the person evaluating them.

The way we had hoped to generate a success test was through the use of Amazon's Mechanical Turk service or by amassing interest on a Twitter feed and determining successful sentences based on likes/retweets. By generating a large enough dataset of reviews, we imagined we would be able to generate heuristics for humor/creativity/inspirability. This is yet to be implemented and applied.

Contextual/Semantic Meaning

Throughout the project we experimented incrementally in trying to make our sentences make more sense, while maintaining their originality. We experimented with the effects of token size on sentence creativity and grammar, and ways to ensure originality (in the end, opting for a bag of words approach). As we got farther and farther along in generating better sentences more frequently, we began to think about how we might enforce semantics. Our naive multi-sentence generation currently works (as discussed earlier) by enforcing a structure and generating a new

sentence based on an overlapping word with the previous one. We spent considerable time looking into more intelligence methods in the area of word and sentence embeddings.

While this approach is beyond what we could accomplish given our timeline, one of our next steps would be to try to implement sentence embedding using long short-term memory networks as described by the work of Hamid Pangi et al: "[Learned using sentence pairs,] sentence embedding can better discover salient words and topics in a sentence, and thus is more suitable for tasks that require computing semantic similarities between text strings."[5] By mapping sentences to semantic vector representations, we could implement a semantic filter based on the cosine distance between the first sentence and the newly generated sentence. We expect the hit rate would improve drastically with semantic meaning.

On a more general note, this project has been a very interesting introduction to NLP and we look forward to exploring this field further.

A Program Trace

Appendix A 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:

$$\begin{array}{c} \\ the \end{array} \begin{array}{c} \begin{array}{cccccccc} the & cat & jumped & on & cake & and & ate & treat \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

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

$$\begin{array}{c} \\ the \\ cat \\ jumped \\ on \\ cake \\ and \\ ate \\ treat \end{array} \begin{array}{c} \begin{array}{cccccccc} the & cat & jumped & on & cake & and & ate & treat \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

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 B explains the setup necessary to start generating startup ideas using our model. Start by cloning the repository here: `https://github.com/cgyulay/startup-generator`. A quick start 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

Appendix C details the contributions of each project participant.

Colton Gyulay: Markov algorithm design and pos tagging
Antuan Tran: Multi-sentence generation and evaluation heuristics
Evan Gastman: Scraping, preprocessing, and evaluation heuristics

Realistically, each of us was involved in all aspects of the project. We wrote the code together, we wrote the paper together, and we chuckled good-naturedly together at our comical startup ideas.

References

[1] Steven Bird, Edward Loper, and Ewan Klein. Natural language processing with python. 2009.

[2] T. De Smedt and W. Daelemans. 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] Deng Palangi. Deep sentence embedding using long short-term memory networks. 2015.

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

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

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