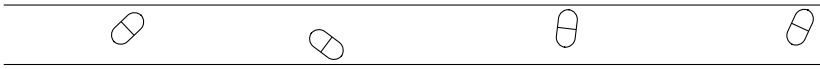

ℙROBABILISTIC 𝕋HINKING

Aryaman Singhal, Kumar Saurav, Lavesh Mangal
8-Paracetamol

April 2023

"Colorless green ideas sleep furiously."
-Noam Chomsky

1 How to generate a natural language?

One of the key tasks of computational linguistics is that of natural language generation(NLG). What then is a natural language? A natural language is a language that has evolved from repetitive use by a human population without conscious planning or premeditation. Natural language is distinct from artificial languages though. Those of you who will take Theory of Computation will get to learn about a fascinating class of languages called *regular languages* that can be modeled by what is called a *"finite state automaton"*. FSAs are a very well-defined mathematical structure, however. Natural language is not so nice.

Suppose then that we are tasked with generating something vaguely resembling natural language.

> *"When an engineer has to build a system to combine words in particular orders, a word-chain device is the first thing that comes to mind."*
> *-Steven Pinker*

The next section will describe how the mentors formulated this problem and implemented the same.

2 Markov Chain Based Text Generation

We describe the process for the English language. The first question you might think of is, is it even computationally feasible to generate English text? Isn't English a rich and complex language? Here are some statistics:

1. According to the Global Language Monitor, English currently has **1 million words**.
2. The average vocabulary of a native English speaker is about **20,000-35,000 words**.

That's not very bad. Most CPUs can execute up to 10^8 instructions in a second. We can even do $O(n^2)$ algorithms in just a couple of seconds, if we work with reasonably small lexicons.

Now, how do we model language?

Formally, given a subset S of English words representing a dictionary, how do we generate a text of length $\approx n$ from it. (The length of a text is defined as the number of words in it. For our purposes, we include punctuation as words.) One way could be as follows:

Algorithm 1 Text Generation Algorithm

Require: Text Length n
  Let s be a sentence.
  while s has less than n words **do**
    Choose a word $w \in_r S$, randomly, uniformly.
    $s \leftarrow s :: w$ ▷ :: here denotes *concatenation*
  end while

Needless to say, this produces absolute gibberish.

So, what is the issue with this approach? The problem here is that English has a syntax, it can model dependencies, it has contexts. In particular, if I stop a sentence mid-way, it is reasonable that one can predict the next-word. In this sense, we can model English as a Stochastic Process.

Consider the set S again. Let $\mathcal{I} = \{1, 2, \ldots, |S|\}$ index the set S. We consider $\mathcal{I}$ to be our state space.

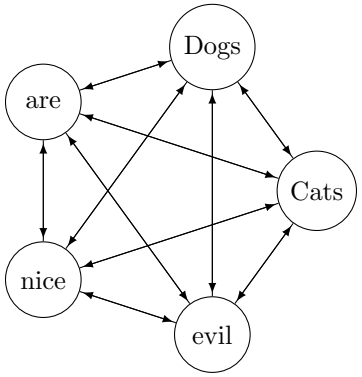

Figure 1: Example Markov Chain that
can generate the sentences:
"Cats are nice.",
"Cats are evil.",
"Dogs are nice.",
"Dogs are evil."

The text-generation task in this sense is the task of simulating the first n random variables in a stochastic process $\{W_j : j \geq 0\}$. We will make use of the following theorem in our simulation process:

Theorem 1. (Inverse Sampling Theorem) *Let $F : \mathbb{R} \to \mathbb{R}$ be a distribution function and let $U \sim Unif(0, 1)$. Then, we define the quantile function also known as the generalized inverse distribution function as follow:*

$$F^{-1}(p) = \inf\{x \in \mathbb{R} : F(x) \geq p\}$$

With the above definition, the random variable $X = F^{-1}(U)$ is distributed as F that is, the distribution function of X is F.

The above theorem is a mathematically precise way of stating a very intuitive fact. Imagine you have a random variable X that takes the values $\{1, 2, 3\}$ with equal probability, then if you obtain a random number $p \in [0, 1]$, you can simulate the random variable as follows:

1. If $p \in \left[0, \frac{1}{3}\right)$, $X = 1$
2. If $p \in \left[\frac{1}{3}, \frac{2}{3}\right)$, $X = 2$
3. If $p \in \left[\frac{2}{3}\right]$, $X = 3$

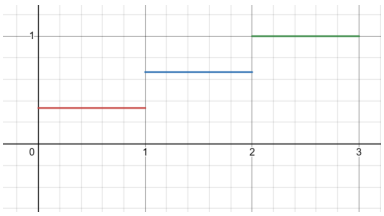

Figure 2: Distribution Function for above example

In this sense, p can be understood to be sampling the value of $F(x)$ and thus the inverse samples from the distribution with DF as F.

Since we have to start somewhere, let us choose the first word randomly i.e.

$$\mathbb{P}(W_0 = i) = \frac{1}{|\mathcal{I}|}, \qquad \forall i \in \mathcal{I}$$

Now, for each of the words $W_1, W_2 \dots W_{n-1}$, we need to simulate the stochastic process according to some distribution. I.e. if we knew the conditional probability mass function $\mathbb{P}(W_j = w | W_{j-1} = w_{j-1}, W_{j-2} = w_{j-2} \dots W_0 = w_0)$, we could generate W_j.

Now, assuming that we have some training data in the form of a corpus, we can *learn* some relationships between words. In particular, if we assume the Markov property, then all we need to do is learn the transition probability matrix P.

We can do so by using the following estimators:

$$p_{ij} = \frac{f_{ij}}{f_i}$$

f_{ij} is the number of occurrences in the text where two successive words are the words indexed by i and j. f_i is just how many times the word indexed by i appears.

Now, supposing that we have learnt the transition probability matrix, how do we simulate the text? Since the support of W_j is S, the cumulative distribution function $F(w) = \mathbb{P}(W_{i+1} \leq w | W_i = w_i)$ is completely specified by its values at each point in S. Now,

$$F(w) = \mathbb{P}(W_{i+1} \leq w | W_i = w_i) = \sum_{j=1}^{w} \mathbb{P}(W_{i+1} = j | W_i = w_i)$$

This corresponds to the sum of the first w terms in the i-th row in the transition probability matrix P.

Thus, we can construct the cumulative distribution function $F(w) = \mathbb{P}(W_{i+1} \leq w | W_i = w_i)$ values as follows, create the prefix sum arrays of each row in the transition probability matrix. We can then find the quantile of any randomly generated number $p \sim \text{Unif}(0, 1)$ using the ith row.

$$F^{-1}(p) = \inf\{x \in \mathbb{R} : F(x) \geq p\}$$

Since the smallest real number for which $F(x)$ takes any real value must be one of the values in the support this is equivalent to:

$$F^{-1}(p) = \min\{i \in S : F(x) \geq p\}$$

This allows us to sample from the corpus' conditional distribution given the last word.

(There is a very nice CF Blog that explains how to find the first element in an array that satisfies some Boolean predicate $f : \mathbb{R} \to \{0, 1\}$ such that $f(x) = (x \geq p)$ using binary search if the boolean predicate is false for some prefix of the array and true for the rest. However, there is a **numpy** function that does exactly this.)

3 Implementation

Here is some general advice for implementing the above:

1. Use Python.
2. You should probably write a class with a generate method and a train method.
3. You can choose whatever corpus of text you like. Public domain texts are easier to find in **.txt** formats. You might find the **nltk** library useful. The mentors recommend Shakespeare, Fitzgerald etc. (Shakespeare has the added advantage that weird text is still funny.)
4. Feel free to use AI tools such as ChatGPT or Bing Chat to help you write your code. It can really help you in writing efficient code for whatever subtask you are trying to achieve. That being said, the purpose of this assignment is to understand the MC structure so try to design it yourself following the hints given above.
5. You may be mildly upset by malformed output. Consider using an NLP tool like spaCy to segment your generated text into sentences and a grammar corrector library such as **gingerit** to correct the text.
6. It is helpful to consider punctuation as words themselves. This indicates a natural sentence delimitation. The last sentence that you generate may have to be discarded because it might not be complete.