# Study the Statistics of the English Language

Pablo Blanco, Aurora Hiveley, Kaylee Weatherspoon

May 1, 2024

**Abstract**

In the English language, certain letters (and even sequences of letters) are more common than others. From a database of all English words, we use Markov chains to construct frequency tables and probability transition matrices for the behavior of the words of the English language English words. Using these tools, we construct "random words" that are likely to be words, but are not. Our analysis of different methods to produce random words, using variables such as word length, method of transition selection, and state size, reveals which random words are qualitatively most similar to true English words.

# 1 Introduction

## 1.1 Markov Processes

To understand Markov Processes, we first introduce the broader category of stochastic processes with a formal definition below.

**Definition 1.** [1] Suppose $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space, and that $I \subset \mathbb{R}$ is of infinite cardinality. Suppose further that for each $\alpha \in I$, there is a random variable $X_\alpha : \Omega \to \mathbb{R}$ defined on $(\Omega, \mathcal{F}, \mathbb{P})$. The function $X : I \times \Omega \to \mathbb{R}$ defined by $X(\alpha, \omega) = X_\alpha(\omega)$ is called a *stochastic process* with indexing set $I$, and is written $X = \{X_\alpha, \alpha \in I\}$.

In simpler terms, a stochastic process is a function $X$ which sends a pair to a real number, where the first element of the pair "chooses" the function $X_\alpha$ from the sample space, $\Omega$, to $\mathbb{R}$. For the sake of intuition, it may be useful

to note that a stochastic process very loosely resembles a group action, in the sense that the set $I$ is acting on $\Omega$.

We often interpret the index set $I$ as a set of times, so that $X_\alpha(\omega)$ is the state of a system at time $\alpha$. This is the case, for example, in the case of *random walks*. In fact, a random walk is an example of a Markov Process, which we define below.

**Definition 2.** [3] A stochastic process has the *Markov Property* if the conditional probability distribution of future states of the process depends only on the present state. If a stochastic process has the Markov Property, we call it a *Markov Process*.

A Markov Process is a *Markov Chain* when either the state space or index set of the process is discrete. In this paper we restrict to Markov Chains, because the set of positions of a letter or string of letters in an $n$-letter word is discrete. There are many continuous Markov Chains of significant interest in the literature, however, with Brownian motion being a famous example.

## 1.2 The Transition Probability Matrix

Given a stochastic process with state space $S = 0, \ldots, D$, we define the transition probability $T^{mn}$ at $\alpha$ to be the likelihood of moving from state $m$ to state $n$ at time $\alpha$. That is, $T^{mn} = \Pr(n|m)$.

**Definition 3.** With $S, D, T$, $m$, and $n$ as above, the *Transition Probability Matrix* is the $(D+1) \times (D+1)$ matrix whose $(i,j)^{th}$ entry is $T^{mn}$.

There are many powerful results in the literature regarding the Transition Probability Matrix. Among the most widely applied is the Chapman-Kolmogorov Equation, which allows us to draw conclusions about a stochastic process at a given time using knowledge about the state of the process at two preceding times.

**Theorem 1.** *(Chapman-Kolmogorov Equation [6]) Let $X = [X_0, X_1, \ldots]$ be a Markov chain with transition matrix $P$. For any $\alpha \in \mathbb{N}$, $P^{(\alpha)}(m,n) = \Pr(X_\alpha = n | X_0 = m)$. Then for any $\alpha \geq 0$ and $\beta \geq 0$,*

$$P^{(\beta+\alpha)}(m,n) = \sum_k P^{(\beta)}(n,k) P^{(\alpha)}(k,m).$$

In the context of language analysis, this could reveal insight like the likelihood of a word beginning with 'a' which has 'b' as its third letter and 'c' as its fourth having the letter 'd' as its seventh. Already, we've obtained intricate information using fundamental known infrastructure surrounding the Transition Probability Matrix.

It is worth noting that the row sums and column sums of the transition probability matrix are all 1, by basic principles of probability. As a result, anything which is true for row-, column-, or doubly stochastic matrices also holds for transition probability matrices. For example, solely because the transition probability matrix is square and row-stochastic, we have that its the largest eigenvalue is 1 ([2]). In the following subsection, we explore more advanced results about Markov Processes and the Transition Probability Matrix.

## 1.3 Advanced Properties of the Transition Probability Matrix and Associated Markov Chain

Two central properties in the study of Markov Chains and the Transition Probability Matrix are reducibility and recurrence. Following the convention of [5], we say a Markov chain is *irreducible* if it has only one communcating class, and is *reducible* otherwise. We define a communicating class formally below:

**Definition 4.** We say two states $s_1, s_2$ of our Markov Chain *communicate* if there is some pair of finite non-negative integers $k, j$ such that it is possible to get from $s_1$ to $s_2$ in $k$ steps and from $s_2$ to $s_1$ in $j$ steps. A *communicating class* is a set of states such that each state communicates with every other state.

**Definition 5.** A Markov chain is *irreducible* if every state communicates with every other state.

In terms of the corresponding graph whose weighted adjacency matrix is the Transition Probability Matrix, this is the condition of being strongly connected:

**Definition 6.** A digraph is *strongly connected* if every vertex can be reached by a directed path beginning at any other vertex.

It is well-known that whether an $n$-vertex digraph is strongly connected can be tested using its adjacency matrix $A$. The digraph is strongly connected if and only if all entries of $B = A + A^2 + \ldots + A^n$ are nonzero. From this we obtain that a Markov Chain with $n$ states is irreducible if and only if $T^1 + T^2 + \ldots + T^n$ has only nonzero entries.

For words of short length, at least, this is easy to check computationally. In fact, we construct the letter-to-letter transition matrices in Section 2. Determining whether a Markov chain is irreducible enables us to access deep results about the stationary distribution, which we define below.

**Definition 7.** A *stationary distribution* of a Markov Chain is some vector $\pi$ such that for $T$ the Transition Probability Matrix, $\pi^\top = \pi^\top T$.

A highly significant parameter, the stationary distribution reveals the long-term behavior of a Markov chain. In context, this would tell us what the distribution of letters would be arbitrarily far from the starting letter. The following result combined with our proof that our small-state transition probability matrices are irreducible gives that we can completely determine the long-range distribution.

**Theorem 2.** *[5] A Markov chain is irreducible if and only if it has a unique stationary distribution.*

Not only does an irreducible Markov chain uniquely determine its stationary distribution; in the long run, the probability distributions converge to the stationary distribution:

**Theorem 3.** *(Ergodic Theorem for Markov Chains, [4])*
*Given an irreducible Markov chain, as the number of steps tends to infinity, the distribution at each state converges to the stationary distribution.*

This result implies that after arbitrarily many steps, the letter distribution "forgets" about the initial condition: the starting letter. From the Perron-Frobenius theorem for Markov Chains, we also have a sense of how the stationary distribution looks, along with information about the spectral radius of the transition probability matrix.

**Theorem 4.** *(Perron-Frobenius for Irreducible Markov Chains, [5]) If $T$ is the transition matrix of an irreducible Markov chain, then*

- *$\lambda = 1$ is guaranteed to be an eigenvalue and a simple eigenvalue*

- *All other eigenvalues of $T$ have $|\lambda| \leq 1$, such that the spectral radius of $T$ is 1.*
- *Upon suitable normalization, the eigenvalue $\lambda = 1$ has a left eigenvector equal to the unique stationary distribution $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)^\top$ and a right eigenvector equal to $\boldsymbol{\eta} = (1, 1, \ldots, 1)^\top$.*

**Theorem 5.** *(Perron-Frobenius for Reducible Markov Chains, [5]) If $T$ is the transition matrix of a reducible Markov chain, then*

- *$\lambda = 1$ is guaranteed to be an eigenvalue*
- *The number of linearly independent eigenvectors with $\lambda = 1$ is equal to the number $r$ of recurrent communicating classes in the Markov chain.*
- *All other eigenvalues have $|\lambda| \leq 1$, such that the spectral radius of $T$ is 1.*
- *There are many choices of left and right eigenvectors for $\lambda = 1$. It is possible to choose $\boldsymbol{\pi}_k$ and $\boldsymbol{\eta}_k$ as a pair of left and right eigenvectors for each of the recurrent communicating classes, where $\boldsymbol{\pi}_k$ is the unique stationary distribution associated with each class and $\boldsymbol{\eta}_k$ is an indicator vector with entry 1 for states in the class and zeros elsewhere.*

The spectral theory of the transition probability matrix is rich, and we refrain from including further detail here. Broadly, the results given above illuminate the value of the transition probability matrix and corresponding Markov chain. Not only do these objects encode rich information about complex systems, they allow us to generate new knowledge, predicting future states of the system. In the following section, we focus our attention on the transition probability matrices associated with words in the English language.

# 2 Methods

We are given an initial procedure `ENG()` which generates a list of lists of all English words of length 1 through 28 (with "antidisestablishmentarianism" being the longest word in the English language.) Our task is to perform statistical analysis of all words in this collection, specifically by studying the frequency with which a transition from letter '$\alpha$' to letter '$\beta$' happens. To do

this, we construct a 28x28 transition matrix representing all possible transitions between the 26 letters of the English alphabet (as well as START and END states.) Then we produce pseudo-words by linking together combinations of letters which occur with high frequency, but do *not* produce real English words.

## 2.1 Transition Matrix

In order to utilize a Markov chain, we must construct a transition matrix from the database of all English words. The procedure `MakeTrM(k,stSize)` outputs the transition matrix constructed by examining all words of length $\geq k$ with a state size of `stSize` (whose default is 1).

We begin with a state size of 1, which means that we consider only transitions between single letters of a word (for example, the transition from '$c$' to '$a$' in the word "cat.") The transition matrix is therefore a 28x28 matrix where the entry in row `i` and column `j` corresponds to the transition from '$i$' to '$j$'.

The first row and column correspond to the START state (i.e. column `i` in row 1 counts the number of words starting with '$i$') and the last is for the END state (the same idea, but for words ending in '$i$.') The entries of the transition matrix are determined by looping over the letters of each word in `ENG()` and incrementing the appropriate matrix entry based upon the letters of the transition. We simultaneously tally the total number of transitions considered by the loop, and we then divide the entire matrix by this total count so that each matrix entry is a relative frequency rather than a raw count.

## 2.2 More Transition Matrices

For an alphabet of size $n$, our $m$-state transition matrix encodes the transition probability from a state $(a_1, \ldots, a_m)$ to another $(a_2, \ldots, a_m, b)$, where each coordinate is an element of $L := \{0, \ldots, n+1\}$ and with a further restriction: for each state, only the first coordinate may be 0 and only the last coordinate may be $n+1$. In this convention, 0 denotes our START state and $n+1$ denotes our END state.

We encoded the $m$-state transition matrix into a table, which is indexed by a state $(a_1, \ldots, a_m)$ and a letter $b$ – since all transitions will be of the form $(a_1, \ldots, a_m) \to (a_2, \ldots, a_m, b)$.

## 2.3 Likely English Words

With the transition matrix in hand, we are able to construct "words" that have a high likelihood of being English words, statistically speaking, but are not actual words. The procedure `generateWord(K,A)` inputs a word length `K` and a starting letter `A` (formatted as a lowercase letter without parentheses or quotes). Note that the transition matrix utilized by this procedure is the result of `MakeTrM(K,1)`.

Starting with `A`, the procedure identifies the letter which most commonly follows `A` by selecting the maximum entry of the row corresponding to `A` in the transition matrix. We then repeat this process to find the most common successor of our newly selected letter, and so on until a word of length `K` has been produced. Lastly, we return the decoy word if it is *not* in the English word collection `ENG()`, otherwise the algorithm returns `fail`. Some example calls and outputs are as follows:

```
> generateWord(5,c);
[c, o, n, g, e]
> generateWord(7,o);
[o, n, g, e, r, e, r]
> generateWord(4,q);
[q, u, n, g]
```

However, this generation procedure is rather deterministic as any given combination of `(K,A)` will produce one particular output. To better study how the statistics of `ENG()` may produce non-English words, we may want to produce a number of different words, perhaps by considering more than one possible successor. Hence, we modify our existing code to produce two functions `generateWordRandom(K,A,R)` and `generateWordRandWted(K,A,R)`.

In each of these new procedures, we still produce a word of length `K` with starting letter `A`. But now, we consider the `R` most common successors of the current letter at each step, where `R` is some number between 2 and 25 (note that `R` = 1 produces the same result as `generateWord`.) The procedure `generateWordRandom(K,A,R)` selects one of the `R` most common successors at random with equal probability, meanwhile `generateWordRandWted(K,A,R)` selects one of the `R` most common successors *based on their weighted frequencies*.

For example, in words of length 5 or more, the transition matrix entry corresponding to `[b,c]` is $\frac{2000}{34903}$ while the entry corresponding to `[b,d]` is

$\frac{1383}{34903}$. Then we would expect the former procedure to select either '$c$' or '$d$' as a successor of '$b$' with equal probability, but the latter procedure should select '$c$' about 1.5x as often as '$d$'. We note that the rational numbers representing each of the `R` possible successors will not necessarily sum to 1, so if we were to check the frequency that each letter is selected from `R` successors, such a frequency would *not* necessarily be equal to that transition's entry in the transition matrix.

```
> generateWordRandom(5,c,1);
[c, o, n, g, e]
> generateWordRandom(5,c,3);
[c, a, n, t, e]
> generateWordRandWted(5,c,3);
[c, o, n, e, r]
```

Some output words, particularly those produced from large `R`, appear to the human eye to be further from real English words, and often were unpronounceable by typical English linguistic rules. For example, one possible execution of `generateWordRandom(5,a,4)` produces `[a,t,r,i,s]`, meanwhile `generateWordRandom(5,a,10)` produces `[a,m,s,p,p]`. This effect is somewhat mitigated by the weighted random selector since the more common (and thus more pronounceable) letter combinations appear with higher frequency. For example, `generateWordRandWted(5,a,4)` produced `[a,l,e,s,t]` while `generateWordRandWted(5,a,10)` produced `[a,r,o,o,n,t]`.

Another method we used to combat the production of unpronounceable words was altering the state size used to define the transition matrix. The details for this construction are outlined in Section 2.2. The new procedure, `generateWordSt(M)`, only has input $M$ for the state-size used in the transition matrix. It generates the first $M-1$ letters as follows: use the $k$-state transition matrix to generate the $k$-th letter $l_k$ according to the transition probabilities of $(0, l_1, \ldots, l_{k-1})$. To generate the first letter, for instance, we check the transition probabilities for our START state.

Unlike previous procedures, we do not prescribe a word length to the generated word and instead let the word end probabilistically. We list a few sample outputs below

```
> generateWordSt(2);
[l, i, s, e, n, t, e]
```

```
> generateWordSt(2);
[t, e, r, s]
> generateWordSt(2);
[h, e, i, n, g, s]
```

# 3 Results

With repeated experimentation using the procedures defined in the previous section, certain patterns emerge. For example, we find that we are more successful in generating "word-like" words when we consider a smaller set of possible successors from the likely successors of any letter. We account for some of these trends and expound on factors affecting them below.

Preliminary observations of certain function calls, particularly utilizing `generateWordRandom` and `generateWordRandWted` with large `R`, showed pseudo-words that qualitatively appeared less like real words than pseudo-words produced with smaller `R`. For example, `generateWordRandom(5,a,4)` produced, in one sample execution, `[a,t,r,i,s]`. Meanwhile, one sample call to `generateWordRandom(5,a,10)` output `[a,m,s,p,p]`. This makes intuitive sense as larger `R` considers a larger number of possible successors at each step, and so the produced word may have a larger number of rarer letter combinations. This effect is somewhat mitigated by the weighted random selector since these rarer combinations are selected with much lower frequency.

However, one drawback of this method is that, with higher frequency, the same letters appear repeatedly in a word as certain letter combinations create a sort of ping-pong effect between those high frequency letters. One such example is that `[t,a,t,a,t]` is produced by `generateWordRandWted(5,t,5)`. Another downside of weighted random successor selection is that the produced word is more likely to be a real English word, and so the procedure fails to return a pseudo-word more often than when the successor is selected at random or deterministically.

Further insight into this repetition can be obtained by considering the spectral radius of the transition probability matrix. From the Perron-Frobenius Theorem for Irreducible Markov Chains (see Section 1), we have that the spectral radius of the transition probability matrix is 1. When the spectral radius is close to 1, the *mixing time*, or time to reach a steady state, is fairly long. That is, the time it takes for the letter choice likelihood at any posi-

tion in a word to resemble the stationary distribution is long. It is therefore unsurprising if not expected that the letter choice may loop, as in the case of `[t,a,t,a,t]`, Of course, this phenomenon is magnified in cases where the most likely letter $x$ to follow a given letter $y$ has the property that $x$ is the most likely letter of follow $y$.

Regarding the issue of pronounceability, we observed that certain generated letter sequences are not found in any pre-existing words. We saw this in `[a,m,s,p,p]` and in the following output:

```
> generateWordRandWted(6,q,20);
[q,u,m,i,t,a]
```

Naturally, there are zero occurrences of the substring `[q,u,m]` in our dictionary of existing English words. To remedy this, we considered using larger state sizes through `generateWordSt`. For state size $M$, we can be certain that the procedure will not generate any words which have substrings of length $M + 1$ which are "illegal" in the English language. As an example, we can be sure that for state size at least 2, there will be no words generated which contain `[q,u,m]` as a substring.

There are several theoretical downsides to consider, though. The first is that larger state sizes shrink the sample size used to approximate the "true" transition probabilities in the language. The second is that for large state sizes, we can expect that our words will deviate from existing words much less; this is partly due to the first point, but also due to the fact that there is a definite upper bound on the length of typical English words. If one picks large state size $M$ then many of the positive transition entries will simply be pre-existing words. In the next sample outputs, we can see some evidence for this analysis.

```
> generateWordSt(2);
[c, a, p, p, e, c, i, d, e]
> generateWordSt(3);
[q, u, o, t, a, t, i, c]
> generateWordSt(4);
[d, i, s, r, u, p, t, u, r, e, s]
```

There was also a concern that generated words were too long, but running large samples of `generateWordSt(2)` and `generateWordSt(3)` indicates that the mean generated word length lies in the interval $(8, 10)$, very roughly

estimated to be around 9. For reference, the average word length in our data is around 8.53.

Word generation by our procedures is not entirely satisfactory, however. Our framework does not take into account that certain word states are only likely when they occur earlier or later in a word. Letter states beginning with "o" (15-th letter in the alphabet) are notorious for this. Below we include some strange sample outputs

```
> generateWordSt(2,15);
[o,u,s]
> generateWordSt(2,15);
[o,c,k,e,r,r,a,g,n,i,c,t,i,o,n,s,e]
```

In the first output, we can speculate that this is a result of the very common "-ous" prefix. For the second output, we see "ock" which is seen most often as a suffix (block, rock, sock, etc.). We speculate that a Markov Chain which includes information on both letter states and letter location may be better suited for word generation.

# 4 Conclusion

Through our original transition probability matrix and the modified constructions which grew from it, we constructed and analyzed words which mimic true English words. We found that choosing from a weighted list of the most likely possible successors at each step lead to words which mimic true words well. That is, implementing `generateWordRandWted` successfully gives pseudo-words, as desired. This is especially true when the set of possible successors (size `R`) is kept large. However, for the unweighted implementation, `generateWordRand`, smaller sets of possible successors yield words of higher quality.

Future work could lead to even more efficient and nuanced generation of pseudo-words. One possible avenue to explore is generating a set of transition probability matrices which depend on the position of the starting letter in the word. In using the same transition probability matrix to choose the next letter at each position in a word, we overlook some of the more subtle structure of the English language. That is, we assume the transition probabilities are the same whether we're at the initial letter or the middle of a word, or the end. It may be the case, however, that the transition from '$n$'

to '$d$', for example, is more likely to occur at the end of a word than in the beginning.

# References


[1] Michael Kozdron. *Lecture 1: The Definition of a Stochastic Process*. URL: https://uregina.ca/~kozdron/Teaching/Regina/862Winter06/Handouts/revised_lecture1.pdf. (accessed: 04.13.2024).

[2] David Mandel. *Markov Chains and Stationary Distributions*. URL: https://www.math.fsu.edu/~dmandel/Primers/Markov%20Chains%20and%20Stationary%20Distributions.pdf.. (accessed: 04.13.2024).

[3] *Markov Chain*. URL: https://en.wikipedia.org/wiki/Markov_chain. (accessed: 04.13.2024).

[4] J.R. Norris. *Markov Chains*. Cambridge University Press, 2012.

[5] Eddie Seabrook, Laurenz Wiskott, and Steven Zucker. "A Tutorial on the Spectral Theory of Markov Chains". In: *Neural Computation* 35 (2023), pp. 1713–1796.

[6] Eklavya Sharma. *Chapman-Kolmogorov Equation*. URL: https://sharmaeklavya2.github.io/theoremdep/nodes/probability/markov-chains/chap-kol-eqn.html#:~:text=Let%20X%20%3D%20%5BX%200%20%2C,called%20the%20Chapman%2DKolmogorov%20equation.. (accessed: 04.13.2024).