# Markov Jabberwocky: *fesh*, *excenture*, and the like

John Kerl

Department of Mathematics, University of Arizona

August 26, 2009

## Lewis Carroll's *Jabberwocky* / *le Jaseroque* / *der Jammerwoch*

*'Twas brillig, and the slithy toves*
*Did gyre and gimble in the wabe;*
*All mimsy were the borogoves,*
*And the mome raths outgrabe.*

*≪Garde-toi du Jaseroque, mon fils!*
*La gueule qui mord; la griffe qui prend!*
*Garde-toi de l'oiseau Jube, évite*
*Le frumieux Band-à-prend!≫*

*Er griff sein vorpals Schwertchen zu,*
*Er suchte lang das manchsam' Ding;*
*Dann, stehend unterm Tumtum Baum,*
*Er an-zu-denken-fing.*

. . .

Many of the above words do not belong to their respective languages — yet look like they *could*, or *should*. It seems that each language has its own periphery of almost-words. Can we somehow capture a way to generate words which look Englishy, Frenchish, and so on?

It turns out Markov chains do a pretty good job of it. Let's see how it works.

# Probability spaces

A probability space* is a set $\Omega$ of possible outcomes** $X$, along with a probability measure $P$ on events (sets of outcomes). Example: $\Omega = \{1, 2, 3, 4, 5, 6\}$, the results of the toss of a (fair) die.

What would you want $P(\{1\})$ to be? What about $P(\{2, 3, 4, 5, 6\})$? And of course, we want $P(\{1, 2\}) = P(\{1\}) + P(\{2\})$.

The axioms for a probability measure encode that intuition. For all $A, B \subseteq \Omega$:

- $P(A) \in [0, 1]$ for all $A \subseteq \Omega$
- $P(\Omega) = 1$
- $P(A \cup B) = P(A) + P(B)$ if $A$ and $B$ are disjoint.

Any function $P$ from subsets of $\Omega$ to $[0, 1]$ satisfying these properties is a probability measure. Connecting that to real-world "randomness" is an application of the theory.

(*) Here's the fine print: these definitions work if $\Omega$ is finite or countably infinite. If $\Omega$ is uncountable, then we need to restrict our attention to a $\sigma$-field $\mathcal{F}$ of $P$-measurable subsets of $\Omega$. For full information, you can take Math 563.

(**) Here's more fine print: I'm taking my random variables $X$ to be the identity function on outcomes $\omega$.

# Independence of events

Take a pair of fair coins. Let $\Omega = \{HH, HT, TH, TT\}$. What's the probability that the first or second coin lands heads-up? What do you think $P(HH)$ ought to be?

| | *H* | *T* |
|---|---|---|
| *H* | 1/4 | 1/4 |
| *T* | 1/4 | 1/4 |

*A* = 1st is heads

*B* = 2nd is heads

Now suppose the coins are welded together — you can only get two heads, or two tails: now, $P(HH) = \frac{1}{2} \neq \frac{1}{2} \cdot \frac{1}{2}$.

| | *H* | *T* |
|---|---|---|
| *H* | 1/2 | 0 |
| *T* | 0 | 1/2 |

*A* = 1st is heads

*B* = 2nd is heads

We say that events $A$ and $B$ are independent if $P(A \cap B) = P(A)P(B)$.

# PMFs and conditional probability

A list of all outcomes $X$ and their respective probabilities is a probability mass function or PMF. This is the function $P(X = x)$ for each possible outcome $x$.

| | | | | | |
|---|---|---|---|---|---|
| 1/6 | 1/6 | 1/6 | 1/6 | 1/6 | 1/6 |

Now let $\Omega$ be the people in a room such as this one. If $9$ of $20$ are female, and if $3$ of those $9$ are also left-handed, what's the probability that a randomly-selected female is left-handed? We need to scale the fraction of left-handed females by the fraction of females, to get $1/3$.

| | L | R |
|---|---|---|
| F | 3/20 | 6/20 |
| M | 2/20 | 9/20 |

We say

$$P(L \mid F) = \frac{P(L, F)}{P(F)}.$$

This is the conditional probability of being left-handed given being female.

# Die-tipping and stochastic processes

Repeated die rolls are independent. But suppose instead that you first roll the die, then tip it one edge at a time. Pips on opposite faces sum to 7, so if you roll a 1, then you have a $1/4$ probability of tipping to $2, 3, 4$, or $5$ and zero probability of tipping to 1 or 6.

A stochastic process is a sequence $X_t$ of outcomes, indexed (for us) by the integers $t = 1, 2, 3, \ldots$: For example, the result of a sequence of coin flips, or die rolls, or die tips.

The probability space is $\Omega \times \Omega \times \ldots$ and the probability measure is specified by $P(X_1 = x_1, X_2 = x_2, \ldots)$. Using the conditional formula we can always split that up into a sequencing of outcomes:

$$\begin{aligned} P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = & P(X_1 = x_1) \\ & \cdot P(X_2 = x_2 \mid X_1 = x_1) \\ & \cdot P(X_3 = x_3 \mid X_1 = x_1, X_2 = x_2) \\ & \cdot P(X_n = x_n \mid X_1 = x_1, \cdots, X_{n-1} = x_{n-1}). \end{aligned}$$

Intuition: How likely to start in any given state? Then, given all the history up to then, how likely to move to the next state?

# Markov matrices

A Markov process (or Markov chain if the state space $\Omega$ is finite) is one such that the

$$P(X_n = x_n \mid X_1 = x_1, X_2 = x_2, \ldots, X_{n-1} = x_{n-1}) = P(X_n = x_n \mid X_{n-1} = x_{n-1}).$$

If probability of moving from one state to another depends only on the previous outcome, and on nothing farther into the past, then the process is Markov. Now we have

$$\begin{aligned} P(X_1 = x_1, \ldots, X_n = x_n) = & P(X_1 = x_1) \\ & \cdot P(X_2 = x_2 \mid X_1 = x_1) \cdots P(X_n = x_n \mid X_{n-1} = x_{n-1}). \end{aligned}$$

We have the initial distribution for the first state, then transition probabilities for subsequent states.

Die-tipping is a Markov chain: your chances of tipping from 1 to $2, 3, 4, 5$ are all $1/4$, regardless of *how* the die got to have a 1 on top. We can make a transition matrix. The rows index the from-state; the columns index the to-state:

$$\begin{bmatrix} & (1) & (2) & (3) & (4) & (5) & (6) \\ (1) & 0 & 1/4 & 1/4 & 1/4 & 1/4 & 0 \\ (2) & 1/4 & 0 & 1/4 & 1/4 & 0 & 1/4 \\ (3) & 1/4 & 1/4 & 0 & 0 & 1/4 & 1/4 \\ (4) & 1/4 & 1/4 & 0 & 0 & 1/4 & 1/4 \\ (5) & 1/4 & 0 & 1/4 & 1/4 & 0 & 1/4 \\ (6) & 0 & 1/4 & 1/4 & 1/4 & 1/4 & 0 \end{bmatrix}$$

# Markov matrices, continued

What's special about Markov chains? (1) Mathematically, we have matrices and all the powerful machinery of eigenvalues, invariant subspaces, etc. If it's reasonable to use a Markov model, we would want to. (2) In applications, Markov models are often reasonable.

Each row of a Markov matrix is a conditional PMF: $P(X_2 = x_j \mid X_1 = x_i)$.

The key to making linear algebra out of this setup is the following law of total probability:

$$\begin{aligned} P(X_2 = x_j) &= \sum_{x_i} P(X_1 = x_i, X_2 = x_j) \\ &= \sum_{x_i} P(X_1 = x_i) P(X_2 = x_j \mid X_1 = x_i). \end{aligned}$$

PMFs are row vectors. The PMF of $X_2$ is the PMF of $X_1$ times the Markov matrix $M$. The PMF of $X_8$ is the PMF of $X_1$ times $M^7$, and so on.

## Back to words! Phase 1 of 2: read the dictionary file

Word lists (about a hundred thousand words each) were found on the Internet: English, French, Spanish, German. The state space is $\Omega \times \Omega \times \ldots$ where $\Omega$ is all the letters found in the dictionary file: *a-z*, perhaps *ô*, *ß*, etc.

After experimenting with different setups, I settled on a probability model which is hierarchical in word length:

- I have $P(\text{word length} = \ell)$.
- Letter 1: $P(X_1 = x_1 \mid \ell)$. Then $P(X_k = x_k \mid X_{k-1} = x_{k-1}, \ell)$ for $k = 2, \ldots, \ell$.
- I use separate Markov matrices ("non-homogeneous Markov chains") for each word length and each letter position for that word length. This is a lot of data! But it makes sure we don't end words with $gr$, etc.

PMFs are easy to populate. Example: dictionary is *apple*, *bat*, *bet*, *cat*, *cog*, *dog*. Histogram:

$$\left[\begin{array}{ccccc} 0 & 0 & 5 & 0 & 1 \\ (\ell=1) & (\ell=2) & (\ell=3) & (\ell=4) & (\ell=5) \end{array}\right]$$

Then just normalize by the sum to get a PMF for word lengths:

$$\left[\begin{array}{ccccc} 0 & 0 & 5/6 & 0 & 1/6 \\ (\ell=1) & (\ell=2) & (\ell=3) & (\ell=4) & (\ell=5) \end{array}\right]$$

# Example

Dictionary is *apple*, *bat*, *bet*, *cat*, *cog*, *dog*. Word-length PMF, as above:

$$\begin{bmatrix} 0 & 0 & 5/6 & 0 & 1/6 \\ (\ell = 1) & (\ell = 2) & (\ell = 3) & (\ell = 4) & (\ell = 5) \end{bmatrix}$$

Letter-1 PMF for three-letter words:

$$\begin{bmatrix} 2/5 & 2/5 & 1/5 \\ (b) & (c) & (d) \end{bmatrix}$$

Letter-1-to-letter-2 transition matrix for three-letter words:

$$\begin{bmatrix} & (a) & (e) & (o) \\ (b) & 1/2 & 1/2 & 0 \\ (c) & 1/2 & 0 & 1/2 \\ (d) & 0 & 0 & 1 \end{bmatrix}$$

Letter-2-to-letter-3 transition matrix for three-letter words:

$$\begin{bmatrix} & (t) & (g) \\ (a) & 1 & 0 \\ (e) & 1 & 0 \\ (o) & 0 & 1 \end{bmatrix}$$

# Phase 2 of 2: generate the words using CDF sampling

How can we sample from a non-uniform probability distribution? Think of the PMF as a dartboard. We throw a uniformly wild dart. Outcomes with bigger $P$ should take up bigger area on the dartboard.

Theorem: This works. Technically:

- We write a cumulative distribution function, or CDF. Whereas the PMF is $f(x) = P(X = x)$, the CDF is $F(x) = P(X \leq x)$. (Put some ordering on the outcomes.)
- Let $U$ (the dart) be uniformly distributed on $[0, 1]$.
- Then $F^{-1}(U)$ (appropriately interpreted) has the distribution we want. (See my September 2007 grad talk *Is 2 a random number?* for full details.)

Example: PMF for letter 1 of three-letter words is

$$\begin{bmatrix} 0.4 & 0.4 & 0.2 \\ (b) & (c) & (d) \end{bmatrix}.$$

CDF for letter 1 of three-letter words is

$$\begin{bmatrix} 0.4 & 0.8 & 1.0 \\ (b) & (c) & (d) \end{bmatrix}.$$

If $U$ comes out to be $0.6329$, then I pick letter 1 to be $c$. If $U$ comes out to be $0.1784$, then I pick letter 1 to be $b$. Etc. I also make a CDF for each row of each Markov matrix.

# Word generation, continued

To generate a word, given the Markov-chain data obtained from a specified dictionary file:

- Use CDF sampling to pick a word length $\ell$ from the word-length distribution.
- Use the letter-1 CDF for word length $\ell$ to pick a first letter.
- Go to that letter's row in the letter-1-to-letter-2 transition matrix for word length $\ell$. Sample that CDF to pick letter 2.
- Keep going until the $\ell$th letter.
- Print the word out.

# Three-letter memory

The non-Markov part of the story: Using Markov chains, as described here, I got decent words, but not always. Real-word correlations go more than one letter deep.

Example: Using a German dictionary, my program generated the 5-letter word *bller*. This made sense: There are *b l* _ _ _ words in German, e.g. *bleib*. There are _ *l l* _ _ words in German, e.g. *alles*. But my Markov model only looks at correlations between adjacent letters, and thus it didn't detect that *bll* _ _ never happens in German.

For revision two of the project, I did all the steps described in the previous slides, but now with the following data:

- I have $P(\text{word length} = \ell)$ as before.
- For first letters, $P(X_1 = x_1 \mid \ell)$.
- For second letters, $P(X_2 = x_2 \mid X_1 = x_1, \ell)$.
- For the rest, $P(X_k = x_k \mid X_{k-2} = x_{k-2}, X_{k-1} = x_{k-1}, \ell)$.

# Results with a tiny word list

Dictionary is *bake*, *balm*, *bare*, *cake*, *calm*, *care*, *cart*, *case*, *cave*. Here are all possible outputs (all of $\Omega \times \Omega \times \ldots$) using two-letter and three-letter memory, respectively. Words appearing in the output but not in the input word list are marked with $*$.

| $\omega$ | $P(\omega)$ | $\omega$ | $P(\omega)$ |
|---|---|---|---|
| *bake* | 0.0740741 | *bake* | 0.1111111 |
| *balm* | 0.0740741 | *balm* | 0.1111111 |
| *bare* | 0.0740741 | *bare* | 0.0740741 |
| *bart** | 0.0370370 | *bart** | 0.0370370 |
| *base** | 0.0370370 | *cake* | 0.1111111 |
| *bave** | 0.0370370 | *calm* | 0.1111111 |
| *cake* | 0.1481481 | *care* | 0.1481481 |
| *calm* | 0.1481481 | *cart* | 0.0740741 |
| *care* | 0.1481481 | *case* | 0.1111111 |
| *cart* | 0.0740741 | *cave* | 0.1111111 |
| *case* | 0.0740741 | | |
| *cave* | 0.0740741 | | |

When larger word lists are used, $\Omega$ is far larger than the input word list: i.e. far more *mimsy* and *mome* than *were* and *the*.

# Results with real word lists

For full-size word lists, I don't try to enumerate all possible outputs — I just generate 100 or so at a time.

When I feed word lists from different languages into the same computer program, I get different outputs. Hopefully, you can tell which is which.

*churency kingling supprotophated doconic linictoxly stewalorties murine hawkinesses*

*texueux roseras plaçâtes exhumèrent orileffé cinquetassions laissiez regre-nèses sauceptant montrenards résaïsmez enjupillâmes ratît fausive*

*perónimo bolón sanfija morricete esmotorrar bisfato filamberecer estempolí mícleta zarífero senestrosia desalificapio*

*Böservolle techtausfälle Nah wohlassee verschützen Probinus träßcher Postenpland einprückt Bußrfere höhegendeter*

*occlamo domitor nestum inhibeo prohisus equino eribro obvolla exteptor exibro abduco loci equa occasco*

# Matching

Aramian Wasielak's idea: run a word (real or not) through the Markov-chain data for all tabulated languages, computing the probability of the word:

$$P(\text{word length} = \ell) \cdot P(X_1 = x_1 \mid \ell) \cdot P(X_2 = x_2 \mid X_1 = x_1, \ell) \cdots$$

(last four columns.) Then, for each word, normalize those numbers to get a score between zero and one (first four columns).

| Word | En score | Fr score | Sp score | De score | En $P$ | Fr $P$ | Sp $P$ | De $P$ |
|---|---|---|---|---|---|---|---|---|
| cat | 1.000 | 0.000 | 0.000 | 0.000 | $5.5 \cdot 10^{-6}$ | 0 | 0 | 0 |
| baguette | 0.015 | 0.985 | 0.000 | 0.000 | $4.7 \cdot 10^{-9}$ | $3.1 \cdot 10^{-7}$ | 0 | 0 |
| wurst | 0.180 | 0.000 | 0.000 | 0.820 | $1.2 \cdot 10^{-7}$ | 0 | 0 | $5.5 \cdot 10^{-7}$ |
| palapa | 0.014 | 0.056 | 0.930 | 0.000 | $9.0 \cdot 10^{-9}$ | $3.6 \cdot 10^{-8}$ | $6.0 \cdot 10^{-7}$ | 0 |
| fesh | 1.000 | 0.000 | 0.000 | 0.000 | $9.3 \cdot 10^{-7}$ | 0 | 0 | 0 |
| location | 0.719 | 0.098 | 0.000 | 0.181 | $1.9 \cdot 10^{-7}$ | $2.6 \cdot 10^{-8}$ | 0 | $4.8 \cdot 10^{-8}$ |
| xyzzy | 0.000 | 0.000 | 0.000 | 0.000 | 0 | 0 | 0 | 0 |
| brillig | 0.000 | 0.000 | 0.000 | 1.000 | 0 | 0 | 0 | $2.5 \cdot 10^{-9}$ |
| slithy | 1.000 | 0.000 | 0.000 | 0.000 | $2.1 \cdot 10^{-7}$ | 0 | 0 | 0 |
| toves | 0.000 | 0.000 | 0.000 | 0.000 | 0 | 0 | 0 | 0 |
| outgrabe | 0.000 | 0.000 | 0.000 | 0.000 | 0 | 0 | 0 | 0 |
| frumieux | 0.067 | 0.895 | 0.000 | 0.037 | $4.5 \cdot 10^{-11}$ | $6.0 \cdot 10^{-10}$ | 0 | $2.5 \cdot 10^{-11}$ |
| griff | 0.742 | 0.139 | 0.000 | 0.118 | $7.4 \cdot 10^{-7}$ | $1.3 \cdot 10^{-7}$ | 0 | $1.1 \cdot 10^{-7}$ |
| vorpal | 1.000 | 0.000 | 0.000 | 0.000 | $1.3 \cdot 10^{-9}$ | 0 | 0 | 0 |
| muggle | 1.000 | 0.000 | 0.000 | 0.000 | $1.5 \cdot 10^{-6}$ | 0 | 0 | 0 |
| expecto | 0.000 | 0.000 | 1.000 | 0.000 | 0 | 0 | $8.1 \cdot 10^{-7}$ | 0 |
| patronum | 1.000 | 0.000 | 0.000 | 0.000 | $2.0 \cdot 10^{-10}$ | 0 | 0 | 0 |

# Other possibilities

In this project, my goal was to construct words out of letters, using language-specific empirical knowledge of transition probabilities from one letter to the next.

One can do something similar, constructing sentences out of (true) words, using language-specific empirical knowledge of transition probabilities from one word to the next. Google for Garkov and Rooter. See also Cam McLeman's page on language/math experiments.

Shane Passon's idea: Using more languages (e.g. German, Dutch, Swedish; French, Spanish, Catalan, Italian; Polish, Czech, Russian; etc.) can we adapt the scoring mechanism to measure relatedness of languages?

All the machinery here works on letters — specifically on written language. Better results might be obtained by using not letters, but units such as $e$, $n$, $ou$, $gh$. This requires a language expert to decide what the pieces are. Or does it? Can we automate detection of these digraphs, trigraphs, and so on?

When we invent nonsense sayings, I don't think there are little Markov chains running in our heads. What's so satisfying about Carroll's *Long time the manxome foe he sought* . . . , and where does it really come from?

Vielen Dank für Ihre Aufmerksamkeit!

Je vous remercie de votre attention!

Gracias por su atención!

Thank you for attending!