A Trigram Model for the Calculation of the Entropy of Printed Portuguese

Carolina Moura

Abstract

I present a novel estimate of an upper bound of 2.00 bits for the entropy of characters in printed Portuguese, obtained by constructing a token trigram model from a large Portuguese corpus and then computing the cross-entropy between this model and a balanced sample of Portuguese text. The higher entropy observed for Portuguese compared to English aligns with expectations from its more complex inflectional morphology and larger space of possible word forms.

1 Introduction

The necessity of talking about "entropy of a language" comes from the desire of understanding how much information is encoded, on average, in each character. For instance, if you're analyzing a piece of English text and see the letter *q*, you know that the next letter will be *u* with high probability, so there is a lot of redundancy when we write *qu*. The amount of information we gain from the next letter given everything we've seen so far puzzled Shannon, who first proposed a few different methods to estimate this number for printed English. Since then, several other studies attempted to tighten the estimate in English, but no calculation more robust than a first-order character approximation was done on Portuguese. In this paper, I reproduce a disruptive study published in ACL 1992 by the IBM T.J. Watson Research Center.

In section 2, I start by modeling language as an ergodic stochastic process and proving that the cross-entropy of any model M' is an upper bound on the entropy rate of the true model M. In section 3, I explain the trigram model as an approximation of the true model and explain tokenization. In section 4, I show concretely how I implemented these ideas. In section 5, I show my results.

2 Language Modeling

2.1 Language as a stochastic process

Every time we speak or write, our choice for the next word depends on both our situational context (who we are, who we're talking to, what idea we want to express) and on the linguistic context (what has already been said). If I started my sentence with "I just ate a", I'm very unlikely to continue with "rock" – unless I'm allowing myself some poetic license or simply holding peculiar habits. However, these two situations are infrequent, so it's reasonable to assume that a word that represents something edible will end my sentence with more likelihood than "rock".

This way, we could approximate my choice for the next word as a distribution $M(\cdot \mid \textit{I just ate a})$. Of course, my distribution M wouldn't be the same as another person's, if we average out the entire world's situational context, we can approximate an universal M – namely, English itself. A more fine-grained approach is to look at characters rather than words. This way, we can view "language" as a discrete stochastic process $\{X_t \in \Sigma : t \in \mathbb{Z}^+\}$ (Shannon, 1948), where Σ is composed of all characters in the alphabet, plus spaces and punctuation. The process is defined by the conditional probabilities $\{p(x_t \mid \mathbf{x_{<t}}) : x_t \in \Sigma, \mathbf{x_{<t}} \in \Sigma^*\}$ (Du et al., 2024), describing the probability of x_t being the next character given the prefix $\mathbf{x_{<t}} = x_1, ..., x_{t-1}$. Modeling language this way gives us a distribution on Σ^*, that is, an associated probability with every sentence that could ever be expressed.

This probabilistic framework distinguishes natural language from pure noise. Under this stochastic process view, English is as an infinite source of symbols, where the next symbol is drawn from a probability distribution that depends on everything that has been drawn before. In the Infinite Monkey Theorem, an idea that has lived in the popular imagination for a long time (see Jorge Luis Borges'

Total Library), a monkey hits a typewriter's keys independently and at random for an infinite amount of time. The monkey will surely, at some point, write the entirety of Hamlet by Shakespeare. In Shannon's model, however, English is a smarter monkey. For English, Hamlet isn't the output of drawing identically and independently distributed events, but a specific, highly probable sequence of characters.

2.2 The entropy rate is well-defined

Let X be our discrete stochastic process. If we have a sequence of n symbols, the amount of information we get from the sequence is $H(X_1, ..., X_n)$. Therefore, the *average* amount of information per symbol is $\frac{1}{n}H(X_1, ..., X_n)$. If we send n to infinity, we have the exact definition of entropy rate (Cover and Thomas, 2006):

$$H(X) = \lim_{n\to\infty} \frac{1}{n} H(X_1, ..., X_n)$$

And this is the value of the average information we get from each character, that is, the value what we want to find. However, it may not exist. Tradionally in the literature, we've assumed language to be an ergodic process. This is far from the truth: the probabilities do change based on the token's position in the sentence (for instance, the first letter is capitalized), so two equal-sized segments don't always hold the same properties. However, for mathematical convenience, this assumption has been adopted, and turns out to be a great approximation, as most modern LLMs assume some sort of local ergodicity or include positional encodings. This way, we'll also model Portuguese as ergodic (and therefore stationary), so the entropy rate is well-defined.

2.3 Cross-entropy as an upper bound

We've proved that, under our model, we can calculate Portuguese's entropy rate. However, we don't know the true underlying distribution P that governs the language. All we can do is try to approximate P by some calculable distribution M, and we'd like to know how different our calculations are with M as if they were with P. The cross-entropy of P as measured by M is defined by:

$$\begin{aligned} H(P, M) &\equiv \lim_{n\to\infty} -E_P \log M(X_n \mid X_{n-1}, ..., X_1) \\ &= \lim_{n\to\infty} -\frac{1}{n} \log M(X_1, ..., X_n) \end{aligned}$$

(Where we got equality by our ergodic and stationary assumptions).

Since cross-entropy is $H(P, M) = H(P) + D(P||M)$, we have by the nonnegativy of the Kullback–Leibler divergence that $H(P) \leq H(P, M)$:

$$\boxed{H(P) \leq \lim_{n\to\infty} -\frac{1}{n} \log M(X_1, ..., X_n)}$$

Since M is ergodic and stationary, then by the Shannon-McMillan-Breiman theorem we can get $-\frac{1}{n} \log M(X_1, ..., X_n)$ by using a sufficiently long sentence drawn according to P. That is, if we have a model M and a representative long sample $x_1, ..., x_n$, we'll have an upper bound on the entropy of the true distribution.

3 Methods

3.1 The trigram model

As presented earlier, the problem reduces to finding an approximation model M and calculating $M(X_1, ..., X_n)$ for reasonably long text (large n). We know:

$$M(X_1, ..., X_n) = M(X_1) \prod_{i=2}^{n} M(X_i \mid \mathbf{X_{<i}})$$

So a strategy is to somehow compute $M(X_i \mid X_{i-1}, ..., X_1)$ for all characters and $i \leq n$. We could read from several samples to calculate these numbers:

$$M(x_i \mid \mathbf{x_{<i}}) = \frac{\#(\mathbf{x_{<i}}, x_i)}{\#(\mathbf{x_{<i}})}$$

For short prefixes, we'll have multiple observations of the same context, so we can estimate a stable conditional probability. However, for long prefixes, the prefix tends to be unique, so we end up having very unstable estimates. For a large prefix, this approach blows up variance (most contexts are unseen), but for too small prefix size, our model increases inaccuracy due to the smaller context. Inspired by Brown et al.'s (1992), we'll use prefix size equals 2. That is, we'll use a *trigram model*:

$$M(\mathbf{X}) = M(X_1, X_2) \prod_{i=3}^{n} M(X_i \mid X_{i-1}, X_{i-2})$$

But isn't the trigram model extremely limited, as it captures almost no context? In their original

paper, Brown et al., 1992 say "linguistically the trigram concept, which is the workhorse of our language model, seems almost moronic [*sic*]". It does seem "moronic", as how can we hope that the two previous words would encode sufficient context? Indeed, modern causal LLMs (like our model is causal) are estimated to achieve, in English, 0.85 bits per byte (Badger and Neligeorge, 2025), which is approximately 0.85 bits per character in English (as 1 character can be encoded in less than 1 byte). Modern causal LLMs take in the whole prefix in order to predict the next token (they might not assign the exact same importance for each token in the prefix, but they definitely do not constraint themselves to the only previous two tokens), which highlights even more the limitations of Brown et al., 1992's method and their 1.75 bits per character result.

Why did they use a trigram model then? Before their study, the most accepted entropy rate for English had been calculated by Shannon, where he asked human subjects to guess the next letter in a text. Brown et al., 1992 inaugarated an era of calculation driven by datasets. This way, I'd like to reproduce this "basic" computation method (a breakthrough at their time).

3.2 Tokenization

Now that we have our computationally feasible model, we need to decide Σ. Notice it would be very meaningless to use trigrams for characters – the model learns very little by knowing $P(e \mid th)$. This way, our estimate of any given sample would be very off, and the cross-entropy would sky-rocket. We could make our grams express more meaningful pieces of text, that is, we have to choose meaningful *tokens*. In a certain way, our choice of tokenizer encodes context beyond the two previous tokens. This is important because language is made of so many subtleties – we have to consider punctuation, whitespaces, diacritics, word roots, inflectional suffixes, etc. It is traditional to create your "token dictionary" (literally an object that maps strings to ids) from the train data and include an *unknown token* to account for any pieces of text we eventually see in the test data that we are unable to fit via our token dictionary.

In Brown et al.'s (1992), they create a token vocabulary that consists of 4 types of tokens:

- **"Standard spelling" token:** they read words from two dictionaries, two name lists (IBM online phone directory and one purchased from a marketing company), a list of places derived from the 1980 US census, and a list of vocabulary used in IBM speech recognition and machine translation experiments.
- **"Punctuation" token:** each punctuation mark was given its own token representation.
- **"Sentence boundary" token:** a special token that is used to separate sentences.
- **"Unknown" token:** represents any eventual spelling not found in the token list.

Summing the amount of each token type, Brown et al.'s (1992) got a vocabulary of size 291,183.

With these tokens, we have several ways of tokenizing a sentence: for instance, if the tokenizer sees "bedrock", should it break it into "bed"+"rock", or just one token "bedrock"? They call each one of the possible ways of tokenizing a sentence a "dissection" of the sentence. This way, the probability of seeing a sentence is composed of the sum of the probability of each dissection:

$$M(\text{string}) = \sum_{\text{dissections}} M(\text{string, dissection})$$

However, computing this value exactly is infeasible, so they end up using a single specific dissection algorithm as representative of the true probability.

I find this approach particularly cumbersome. Corrupted by post-"scalable neural networks" mentality of nowadays (the mentaility that everything is a trainable task), I'll allow myself a "smarter tokenizer". As I've given the intuition, our choice of tokens should somehow encode meaningful pieces of semantics and grammar. Dissecting a sentence into meaningful pieces requires a linguistic study of the language, that is, it requires *knowledge*. If you're creating a rule-based tokenizer (like Brown et al.'s (1992) did, by deciding that punctuation, whitespaces and words were the basis of meaning), these rules turn out to be heuristics. Since I don't want to get into the linguistics of the problem, we can use an ML model to learn these dimensions instead.

In my methods, I used the Tucano tokenizer, that I will describe in 3.2.2. Before that, however, I'd like to discuss the SentencePiece tokenizer, the tokenizer superclass that Tucano belongs to.

3.2.1 SentencePiece

SentencePiece is a language-agnostic subword tokenizer and detokenizer developed by Google researchers Taku Kudo and John Richardson in 2018 (Kudo, 2018). It is an unsupervised model, that is, it requires no human annotation – you just feed it with a large dataset and it finds patterns on its own. When trained with unigram tokenization, its output is a set $V = \{(x, p)\}$ of tokens with their associated probability. What does this mean? Why is the model outputing probabilities? This is because the notion of dissection appears again. Even if you have a vocabulary of tokens, there are always multiple ways of dissecting an input:

$$V = \{\text{'bed'}, \text{'rock'}, \text{'bedrock'}\}$$

$$\text{tokenize('bedrock')} = [\text{'bed'}, \text{'rock'}] \text{ or } [\text{'bedrock'}]$$

If we had the probability $p : V \to [0; 1]$ of each token appearing in English after taking into account all possible tokenizations of all possible English sentences, then we can introduce the notion of "most probable tokenization":

$$P(x_1, \cdots, x_n) = \prod_{i=1}^{n} p(x_i)$$

$$x^* = \underset{\mathbf{x} \in S(X)}{\operatorname{argmax}} P(\mathbf{x})$$

Where $S(X)$ is the set of all possible dissections for a sentence X.

The way that unigram tokenization is implemented is: we start with a tentative V that is way bigger than our desired vocabulary size (maybe all possible substrings in the input) and assign probabilities according to our input (maybe the frequency of each token in the input). See figure 1 for an example of initialization where our only training datum is the sentence "Essa pessoa".

Then, iteratively, it uses the distribution V to calculate the probability of the input. Notice that we could have several dissections, but the original SentencePiece paper claims that the most likely dissection approximates the sum of all dissections very well (i.e., $P(\text{data}) \approx P(\text{best dissection})$. Notice that our goal now is to decrease the size of V, so we have to use this probability estimate to help us achieve this.

SentencePiece then calculates a "loss" for each token: if we were to remove this token from our vocabulary, how much would the input's probability change? Going back to our example:

$$D = \{\text{'Essa pessoa'}\}$$

V	p
' '	1/66
' p'	1/66
...	...
'ss'	1/33
'sso'	1/66
'ssoa'	1/66

Figure 1: Example of how SentencePiece could initialize its state when "Essa pessoa" is the only sample in the training set. Notice how V contains all possible substrings with their probability, the latter matching the frequency (for instance, "ss" appears twice, so its probability is $2/66 = 1/33$). Notice there are $\binom{11}{2} + 11 = 66$ substrings.

Vocab	Best Segmentation	Prob.
V	["Essa pessoa"]	$1/66$
$V \setminus \{\text{"Essa pessoa"}\}$	["Essa pesso", "a"]	$2/66^2$

Table 1: Impact of removing the token "Essa pessoa" from the vocabulary.

For example, the *loss* for the token "Essa pessoa" would be:

$$\begin{aligned}\Delta\mathcal{L}(\text{"Essa pessoa"}) &= \frac{1}{66} - \frac{2}{66^2} \\ &= \frac{16}{1089} \\ &\approx 0.01469\end{aligned}$$

Once we have loss_i for each token $i \in V$, then we sort these losses and get rid of the n% biggest losses, where n% varies by implementation. We stop these iterations when our vocabulary reaches our desired size.

3.2.2 The Tucano tokenizer

In my methods, I used the Tucano tokenizer (Tucano et al., 2025), trained via SentencePiece with unigram tokenization (Kudo and Richardson, 2018) for a family of models that outperform other Portuguese and multilingual language models in several Portuguese benchmarks (Corrêa et al., 2025). Table 2 contains some sample tokenization.

A good thing about the Tucano tokenizer is that it is trained in Portuguese, so the structural patterns it learns are language-specific. It has been showed that language-specific tokenizers potentially provide better perfomance in downstream tasks (Seo et al., 2025).

Text	Translation	Tokenization
Essa pessoa.	That person.	[5841, 20506]
Essa pessoa	That person	[5841, 3271]

Table 2: Tucano's tokenizer sample of tokenization. The vocabulary size is 32,000 tokens. Notice how adding a period at the end of the sentence actually changes the last token, instead of adding a token.

In our tokenization, we're outputing the most probable dissection x^*. Not doing all the dissections is, indeed, still a problem with my tokenization. However, it can only bring more looseness into the upper bound:

$$\begin{aligned} & M(\text{string}) \geq M(t_1, \ldots, t_k) \\ \Rightarrow\ & \log M(\text{string}) \geq \log M(t_1, \ldots, t_k) \\ \Rightarrow\ & -\log M(\text{string}) \leq -\log M(t_1, \ldots, t_k) \end{aligned}$$

Where $t_1, ..., t_k$ is the tokenization by Tucano. So the upper bound can only get worse. I believe using an ML model compensates the usage of a single dissection due to the knowledge it expresses about the language.

3.3 Computing the model

Finally, we just need to calculate:

$$M(t_3 \,|\, t_1 t_2) = \frac{\#(t_1 t_2 t_3)}{\#(t_1 t_2)}$$

$$M(t_1 t_2) = \frac{\#(t_1, t_2)}{\sum_{i,j} \#(t_i t_j)}$$

For some training data. I used Crespo et al.'s (2023), a constantly maintained dataset for Portuguese built by C4AI, a research center for artificial intelligence inside the University of Sao Paulo. 10% of the data was separated for testing, and the remaining 90% for training.

3.4 The smoothing function

When we calculate $M(x_1, ..., x_n)$ for the validation, it can happen that $M(x_i \,|\, x_{i-2} x_{i-1})$ wasn't seen in the training data. If we continued with our model, we would be assigning a zero probability to the sentence. Therefore, we need a smoothing function. In Brown et al.'s (1992), they use Jelinek-Mercer smoothing (Jelinek and Mercer, 1980). This smoothing function is a linear interpolation of the trigram, bigram and unigram model. For each pair of tokens (t_1, t_2), it calculates constants $\lambda_i(t_1 t_2)$ and assigns $M(t_3 \,|\, t_1 t_2)$ to be:

$$\begin{aligned} M(t_3 \,|\, t_1 t_2) = & \lambda_3(t_1 t_2) \cdot f_3(t_3 \,|\, t_1, t_2) \\ & + \lambda_2(t_1 t_2) \cdot f_2(t_3 \,|\, t_2) \\ & + \lambda_1(t_1 t_2) \cdot f_1(t_3) \\ & + \lambda_0(t_1 t_2) \cdot f_0 \end{aligned}$$

In this equation, f are the empirical conditional probabilities calculated by the training data. The λ values are learned by the smoothing model. In their paper, they do not describe the precise way in which they learn the parameters, but it is mentioned that they divide the training data into the main segment and a small "held-out" section. In this "held-out" section, they use the conditional probabilities learned by the main segment, and try to find λ that maximizes the likelihood of the input.

Due to the obscurity of their method, I opted for an additive smoothing:

$$M(t_3 \,|\, t_1 t_2) = \frac{1 + \#(t_1 t_2 t_3)}{|V| + \#(t_1 t_2)}$$

Where $|V| = 32k$ is the size of the token vocabulary. It's unclear to me the magnitude of the role that the choice of smoothing function plays. I believe the simplicity of the additive smoothing compensates the cognitive cost of Jelinek and Mercer's smoothing, but a future study may apply the latter for comparison.

4 Implementation

The implementation can be found at this repository. There are different scripts:

- find_trigrams.ipynb: loads the carolina corpus and calculates unigram, bigram and trigram frequency for 90% of the samples. Since there are 2,108,999 samples (3.1GB) in the corpus, each gram frequency's table consumed 298KiB, 442MiB and 8.5GiB of disk memory, respectively. This way, they were dumped into a ClickHouse database for downstream processing. The script ran for 3h28min in a machine with 51GB of RAM.

- calculate_entropy.py: calculates the character entropy rate in the remaining 10% of the samples. For the script to run fast, I loaded the model to memory. This required a VM with 125GB of RAM, and ran for 2h27min.

Approximation	Entropy rate
Zero-order character	≈ 5.25
First-order character	≈ 3.30
Third-order word	≈ 2.00

Table 3: Comparison between entropy-rate calculations.

- causa_llm.ipynb: loads TeenyTinyLama for evaluating character entropy rate in test data. Ran in an A100 for 4h09min. See discussion about open-source comparison in 5 to understand.

5 Results

We found an average of 2.003 information bits per character when considering a long sample of $806,713,826$ characters. This is expectedly larger than the 1.75 bits found for English by Brown et al.'s (1992), since Portuguese has many verb conjugations, gendered nouns/adjectives, and more unique word forms in general.

Table 3 compares my results with different approaches that take into consideration less context, therefore providing with worse results. Zero-order character approximation (symbols are independent and equiprobable) calculates simply $\log 38$, where 38 is the approximate size of Portugue's alphabet counting diacritics and special characters. First-order drops the equiprobable assumption, which decreases entropy (notice the uniform distribution, the one assumed by zero-order, is the most entropic distribution). The first-order character approximation was also done with the carolina corpus.

Out of curiosity, I tried to run the same test split from the carolina corpus against an open-source causal LLM. For that, I used TeenyTinyLlama (Corrêa et al., 2024), made by the same group that produced the Tucano tokenizer. With that model and my test set, I got a shocking result of 0.505 bits per character. After 7 hours of training, I found out TeenyTinyLlama had been trained with the carolina corpus. I tried to redo the experiment with a dataset that curated 2GB of Portuguese Wikipedia pages, getting a more reasonable result of 0.89 bits per character. Since I don't know if TeenyTinyLlama was trained on these pages, I cannot hold this result as scientific evidence, but it's curious nevertheless.

6 Conclusion

In this work, I proposed an empirical estimate for an upper bound on the entropy rate of printed Portuguese, derived from a token-based trigram model trained on a large contemporary corpus. The estimated bound of 2.00 bits per character indicates that Portuguese exhibits higher informational density than English, reflecting its morphological richness, inflectional complexity, and greater lexical variety. This result supports the long-held linguistic intuition that languages with more productive morphology tend to distribute information across a broader set of word forms, thereby reducing predictability at the character level (Bentz et al., 2017). Notice, however, this is not the same as "language complexity": in Linguistics, the "complexity" is defined as a mix of morphology and syntax, and there is large agreement within the scientific community about the "equi-complexity hypothesis". It seems that if a language is considered less morphologically complex, it would compensate by being more syntatically complex (Bentz et al., 2023).

While the trigram model provides a computationally feasible approximation, it remains a simplification of the true linguistic process. This project's goal was simply to reproduce an important study in the history of Computational Linguistics and Information Theory.

Overall, this study offers a first robust upper bound for Portuguese's printed entropy, mixing classical information-theoretic approaches with modern language modeling.

References

Benjamin L. Badger and Matthew Neligeorge. 2025. Know your limits: Entropy estimation modeling for compression and generalization. *Preprint*, arXiv:2511.10618.

Christian Bentz, Dimitrios Alikaniotis, Michael Cysouw, and Ramon Ferrer i Cancho. 2017. The entropy of words - learnability and expressivity across more than 1000 languages. *Entropy*, 19:275.

Christian Bentz, Ximena Gutierrez-Vasques, Olga Sozinova, and Tanja Samardžić. 2023. Complexity tradeoffs and equi-complexity in natural languages: a meta-analysis. *Linguistics Vanguard*, 9(s1):9–25.

Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. 1992. An estimate of an upper bound for the entropy of english. *Comput. Linguist.*, 18(1):31–40.

Nicholas Kluge Corrêa, Sophia Falk, Shiza Fatimah, Aniket Sen, and Nythamar De Oliveira. 2024. Teenytinyllama: open-source tiny language models trained in brazilian portuguese.

Nicholas Kluge Corrêa, Aniket Sen, Sophia Falk, and Shiza Fatimah. 2025. Tucano: Advancing neural text generation for portuguese. *Patterns*, page 101325.

Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory 2nd Edition (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience.

Maria Clara Ramos Morales Crespo, Maria Lina de Souza Jeannine Rocha, Mariana Lourenço Sturzeneker, Felipe Ribas Serras, Guilherme Lamartine de Mello, Aline Silva Costa, Mayara Feliciano Palma, Renata Morais Mesquita, Raquel de Paula Guets, Mariana Marques da Silva, Marcelo Finger, Maria Clara Paixão de Sousa, Cristiane Namiuti, and Vanessa Martins do Monte. 2023. Carolina: a general corpus of contemporary brazilian portuguese with provenance, typology and versioning information. *Preprint*, arXiv:2303.16098.

Li Du, Holden Lee, Jason Eisner, and Ryan Cotterell. 2024. When is a language process a language model? In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11083–11094, Bangkok, Thailand. Association for Computational Linguistics.

Fred Jelinek and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In Edzard S. Gelsema and Laveen N. Kanal, editors, *Proceedings, Workshop on Pattern Recognition in Practice*, pages 381–397. North Holland, Amsterdam.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. *Preprint*, arXiv:1804.10959.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Jean Seo, Jaeyoon Kim, SungJoo Byun, and Hyopil Shin. 2025. How does a language-specific tokenizer affect llms? *Preprint*, arXiv:2502.12560.

Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423.

Tucano, Nicholas Kluge Corrêa, Aniket Sen, Sophia Falk, and Shiza Fatimah. 2025. Gigaverbo (revision 3464568).