AI in Ancient Rome

Using mathematical models to answer the question: Would NLP Models have been more successful in Ancient Rome?

At first, you might not see the inextricable link between mathematics and language, and if you asked a secondary school pupil, they probably wouldn't have guessed any link between the two existed, but in this essay, I will use mathematical models to explore the pros and cons of training natural language processing (NLP) models on Latin and English, with the speed at which they would be able to learn the language the measure of success.

Introduction[1]

The best AI chatbots such as ChatGPT, the "highest-ranking tool" accounting for "82.5% of all AI tool website traffic" and "used by 400 million people a week", have become so ubiquitous that we don't even marvel at the sheer feat of engineering that they really are anymore.

In this essay I want to draw particular attention to the tool which all great chatbots, such as ChatGPT, use to enhance themselves from simple, rule-based platforms to dynamic, context-aware engagement systems: natural language processing (NLP).

So how could Latin possibly offer a better language to train NLP models on than English?

Well on the one hand, Latin is a highly structured language, whilst English is notoriously full of irregularities and ambiguities.

But on the other hand, Latin has an increased morphological complexity and far less training data.

How an NLP model works[2]

[1] Schwartz, E.H. (2025). *OpenAI confirms 400 million weekly ChatGPT users - here's 5 great ways to use the world's most popular AI chatbot*. [online] TechRadar. Available at: https://www.techradar.com/computing/artificial-intelligence/openai-confirms-400-million-weekly-chatgpt-users-heres-5-great-ways-to-use-the-worlds-most-popular-ai-chatbot

AI (2024). *OpenAI's ChatGPT boasts highest visits among world's most popular AI tools*. [online] Aa.com.tr. Available at: https://www.aa.com.tr/en/science-technology/openai-s-chatgpt-boasts-highest-visits-among-world-s-most-popular-ai-tools/3363105

[2] Hong, Z. (2023). *Attention Mechanisms in Deep Learning: Enhancing Model Performance*. [online] Medium. Available at: https://medium.com/@zhonghong9998/attention-mechanisms-in-deep-learning-enhancing-model-performance-32a91006092a

About to follow is a whistle-stop tour of the workflow of NLP models, which may leave you with many more questions by the end as I am simply detailing it to provide some context for the demos I will provide later.

NLP models typically work through a three-stage process:

1. Data collection - The first step involves gathering text or speech data from a variety of sources such as cloud data warehouses, surveys, emails, or online platforms. For my demos, the text was collected from online e-text libraries like the Gutenberg Project and the Perseus Digital Library, as well as news outlets such as The Economist, CNN, and the Financial Times.

2. Text pre-processing - Next, the raw text is cleaned and prepared using various preprocessing techniques, including:
   - *Tokenization*: Breaking down text into smaller units called tokens (words, parts of words, or individual characters).
   - *Stemming and Lemmatization*: Reducing words to their root form. Stemming uses a rule-based approach, while lemmatization uses linguistic analysis to find the correct dictionary form of a word.
   - *Stop Word Removal*: Removing commonly used words such as "the," "is," and "a" that don't carry much meaning on their own. This helps the model focus on more meaningful words and improves its ability to analyse text.

   After these steps, the text is typically transformed into a structured format suitable for machine learning. One important step here is converting tokens into numerical vectors, or embeddings. These embeddings help the model understand the meaning and relationships between words in a mathematical way — for example, recognizing that "king" and "queen" are related concepts.

3. Training and Application - Once the data is pre-processed and embedded, it is used to train NLP models to perform specific tasks such as sentiment analysis, topic classification, translation, or question answering. Training involves feeding tons of data into the model so it can learn patterns and improve accuracy. Modern NLP models, such as BERT and GPT, use a transformer architecture (a deep learning model that uses a self-attention mechanism), which is particularly effective at understanding context within and across sentences. These models are typically pretrained on massive datasets before being fine-tuned for specific tasks, e.g. sentiment analysis, using smaller, domain-specific datasets.

What do NLP models do well at and what do they struggle with?

Strengths: text generation, summarisation, sentiment analysis, translation, chatbots

Weaknesses: understanding context and nuance, reasoning and logic, handling ambiguity

---

DeepLearning.AI (2023). *Natural Language Processing (NLP) - A Complete Guide*. [online] www.deeplearning.ai. Available at: https://www.deeplearning.ai/resources/natural-language-processing/

Language traits that affect learning speed

The natural language on which NLP models are trained on have significant impacts on the speed at which the model can pick up the language due to a wide range of reasons such as the structural traits, complexity and unpredictability of a language.

Listed below are the three inherent language traits I've chosen to investigate which would have a significant effect on the speed at which a model would learn a language:

1. Entropy of Language
2. Word frequency distributions
3. Morphological Structure and Inflections

Latin vs English: A structural comparison (Modelling time)

To assess whether Latin would pose an advantage to NLP models being able to learn human language more quickly, I will use three mathematical models to compare these three inherent language traits as detailed above.

To carry out the necessary calculations, I have pulled English and Latin texts from online libraries and sources, and collated the texts into one corpus, in their respective languages, so that the wordcount in each file is approximately 250,000.

The Latin articles were selected such that they encapsulated works of prose, verse and authors from different eras (featured authors included Caesar, Catullus, Cicero, Horace, Ovid, Seneca and Virgil). Similarly, the English texts were chosen so that they covered a wide range of different writing styles, through purpose and time (featured authors and pieces included JK Rowling, the author of Beowulf, a CNN news article, Charles Dickens, Jane Austen, and Shakespeare).

1. Entropy of Language[3]

The first aspect of language I want to analyse is the entropy. The notion of entropy originally comes from physics where it measures the disorder or randomness of a certain system. In language, entropy measures the uncertainty, the lower the entropy, the more predictable the language.

The entropy of language can be split into two subcategories: the entropy of meaning, and the entropy of conversation. Put very shortly, the entropy of meaning measures the uncertainty of meaning arising due to polysemous words and the coverage of meaning

---

[3] Lo, J. (2018). *The Entropy of Language - Language Insights - Medium*. [online] Medium. Available at: https://medium.com/language-insights/the-entropy-of-system-life-and-language-43d89c0d185b

www.sciencedirect.com. (n.d.). *Shannon Entropy - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/engineering/shannon-entropy

The National Archives (2019). The National Archives. *The Cultural Revolution*. [online] doi:https://www.nationalarchives.gov.uk/

associated to a certain word, whilst the entropy of conversation deals with word order and gender limiting the number of possibilities of words which can follow a previous one.

Rigid word order in English, with strong dependencies between certain words (articles → nouns → verbs) means that the next word is more predictable and therefore predicting the next word for an NLP model becomes a simpler task, as it heavily relies on structure and logic.

By contrast, the fact that Latin is a language with a highly inflected morphology, makes it more predictable as the structure of the words gives more clues about the meaning. And moreover, the more limited set of root words and structure present in Latin leads to a smaller variety of possible word combinations and thus a lower entropy.

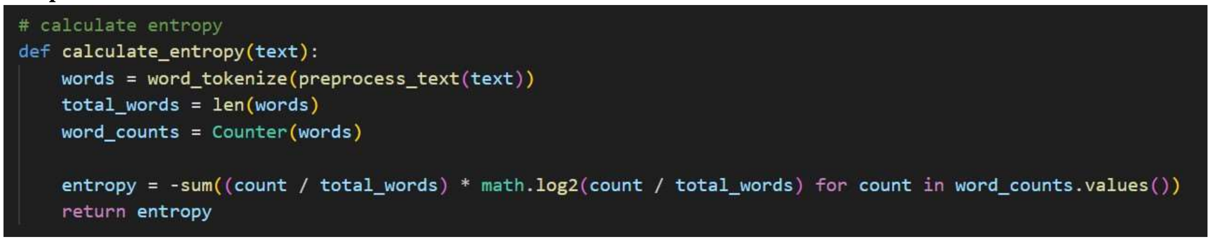
Therefore, we would expect Latin to have a lower net entropy than English when we carry out these calculations.

Model calculation:

This component of a language can be measured by using the Shannon Entropy Formula:

$$H = -\Sigma p(x) log p(x)$$

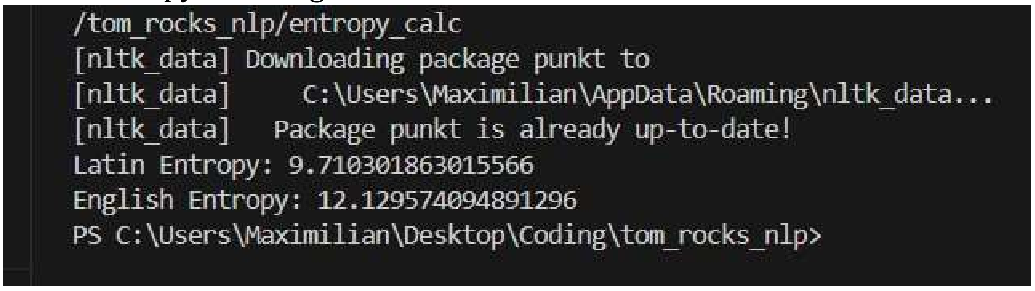
And by using the python code below I assessed the entropy of the Latin and English corpora.



```python
# calculate entropy
def calculate_entropy(text):
    words = word_tokenize(preprocess_text(text))
    total_words = len(words)
    word_counts = Counter(words)

    entropy = -sum((count / total_words) * math.log2(count / total_words) for count in word_counts.values())
    return entropy
```

When I ran this code, it displayed what we had expected: that Latin had a significantly lower entropy than English:



```
/tom_rocks_nlp/entropy_calc
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Maximilian\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Latin Entropy: 9.710301863015566
English Entropy: 12.129574094891296
PS C:\Users\Maximilian\Desktop\Coding\tom_rocks_nlp>
```

Therefore, with regards to entropy, Latin seems to be a better language to train NLP models on than English due to its increased predictability.

2. Zipf's Law – word frequency distributions[4]

[4] Encyclopedia Britannica. (n.d.). *Zipf's law | probability*. [online] Available at: https://www.britannica.com/topic/Zipfs-law

The second aspect of language I wanted to investigate was the word frequency distributions of the respective languages using Zipf's law.

This is an important part of language as Zipf's law states that in any natural language the word frequency *f* of a certain event is inversely proportional to its rank *r*, meaning a small number of words are extremely common, whilst most words are much more scarcely used.

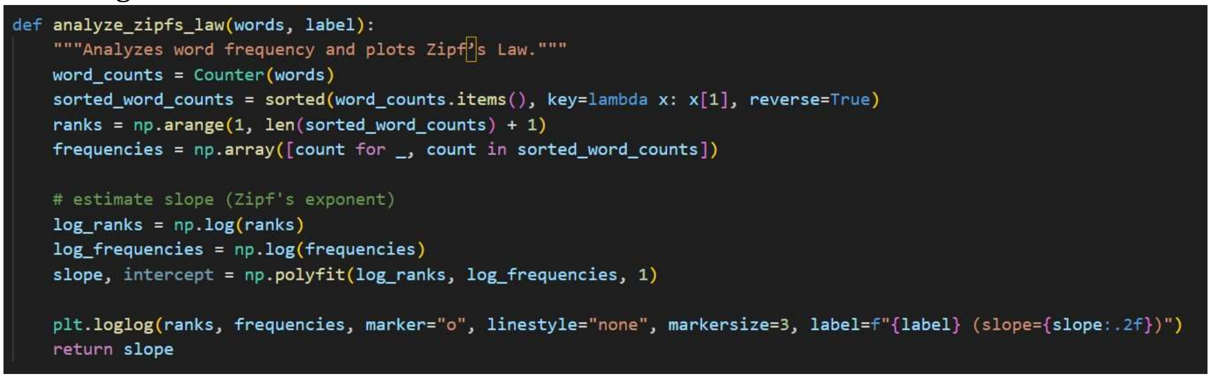
$$f(r) \approx \frac{C}{r^2}$$

$$where\ f(r) = frequency\ of\ word\ at\ rank\ r$$

Word frequency distribution is an important measure to gauge how easily an NLP model would be able to learn a language as a language with a steeper Zipfian distribution on a log-log plot (where common words are even more dominant) would be easier for NLP models to learn initially, as they can leverage high-frequency words more effectively.

Model calculation:

I compared the word frequency distribution on the Latin and English corpora using the following code:

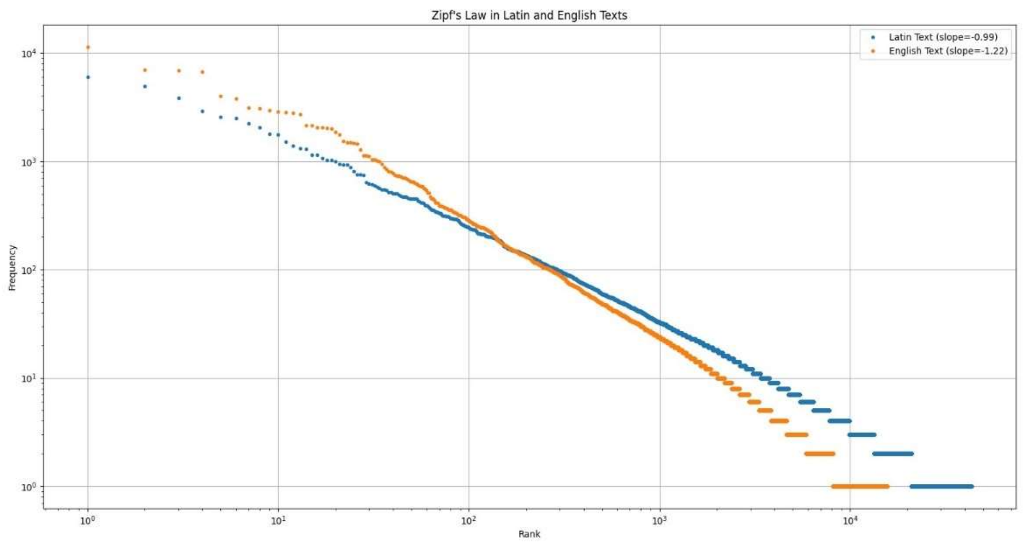


```
def analyze_zipfs_law(words, label):
    """Analyzes word frequency and plots Zipf's Law."""
    word_counts = Counter(words)
    sorted_word_counts = sorted(word_counts.items(), key=lambda x: x[1], reverse=True)
    ranks = np.arange(1, len(sorted_word_counts) + 1)
    frequencies = np.array([count for _, count in sorted_word_counts])

    # estimate slope (Zipf's exponent)
    log_ranks = np.log(ranks)
    log_frequencies = np.log(frequencies)
    slope, intercept = np.polyfit(log_ranks, log_frequencies, 1)

    plt.loglog(ranks, frequencies, marker="o", linestyle="none", markersize=3, label=f"{label} (slope={slope:.2f})")
    return slope
```

To obtain the following log-log graph:



To be able to compare the distributions of Latin and English words, plotting a log-log graph as shown above is accepted practice, as it displays a linear relationship from which we can interpret the gradient of each line respectively to draw a conclusion.

Description of graph:

On this log-log graph the Latin texts clearly show a shallower gradient than the English texts. (-0.99 vs -1.22)

Results interpreted:

Reasons for this shallower slope include the fact that Latin is a highly inflected language (words change form based on case, number, gender, tense, etc).

As a result, this creates more unique word forms which spreads out the word frequency more evenly. Conversely, English relies more on word order and auxiliary words rather than inflections, concentrating the frequency more, particularly auxiliary verbs such as “be”, “have”, “do”, “can”, “will” and “shall”.

Because multiple inflected forms of a single word appear separately, the top ranked words in Latin aren’t as dominant, resulting in a less steep log-log plot.

Coming back to NLP models, a steeper Zipfian slope means that a small number of words dominate the dataset, allowing the model to see these words frequently and learn them effectively. By comparison, Latin’s shallower slope results in frequency being spread across many word forms, which can lead to data sparsity issues in training NLP models, as the model encounters many inflected forms with lower frequency.

3. Demonstrating Transition Probabilities (Markov Chains)[5]

Transition Probabilities show how often one word follows another (the probability of a certain bigram occurring). So, by looking at the differences of these probabilities in Latin and English, we can compare the word predictability, which will affect how quickly a model learns a language.

The transition probabilities in inflected languages (such as Latin) will often reflect how (much) inflectional morphology (changes in word endings) affects how words interact.

For example, in English, with relatively simpler morphology, the transition probability between certain word pairs may be higher because word forms (e.g., "dog", "dogs") do not change much except for pluralization or tense. This means that, for different uses of context of the same word "dog", it can be followed by a wider range of different possibilities.

---

[5] Readthedocs.io. (2016). *Transitional Probability — Phonological CorpusTools 1.5.1 documentation*. [online] Available at: https://corpustools.readthedocs.io/en/latest/transitional_probability.html
Kandala, A. (2023). *From Zero to Hero: Laplace Additive Smoothing for Naive Bayes Classifier.* [online] Medium. Available at: https://anupamakandala1111.medium.com/from-zero-to-hero-laplace-additive-smoothing-for-naive-bayes-classifier-40828d35c6a7

Collins Latin dictionary and grammar. (2016). London: Collins.

By contrast, in Latin, the transition between words like "amare" (to love) and "amavit" (he/she/it loved) will be influenced by inflectional endings and will depend on their morphological forms rather than word order. This means that since each word is a lot more specific about its use within a context, there is a narrower range of different possibilities of words with which it can be followed.

Therefore, when we do this demo, we would expect to see smaller transition probabilities for Latin bigrams.

Model calculation:

So, once the English and Latin corpora have been pre-processed, the bigrams are generated and counted. Then the transition probabilities for each of the top 1000 bigrams are calculated using this formula:

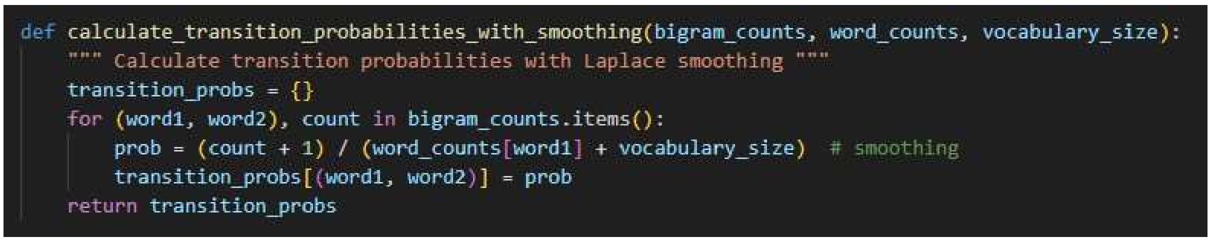
$$p(\mathrm{w}_2 \mid \mathrm{w}_1) = \frac{\mathrm{Count}(\mathrm{w}_1,\mathrm{w}_2)}{\mathrm{Count}(\mathrm{w}_1)}$$

Which I then adjusted to implement a Laplace smoothing, changing the formula to this:

$$p(\mathrm{w}_2 \mid \mathrm{w}_1) = \frac{\mathrm{Count}(\mathrm{w}_1,\mathrm{w}_2)+1}{\mathrm{Count}(\mathrm{w}_1)+\mathrm{V}}$$

$$where\ V\ is\ the\ vocabulary\ size\ (number\ of\ unique\ words)$$

This can be seen in the code below:



```python
def calculate_transition_probabilities_with_smoothing(bigram_counts, word_counts, vocabulary_size):
    """ Calculate transition probabilities with Laplace smoothing """
    transition_probs = {}
    for (word1, word2), count in bigram_counts.items():
        prob = (count + 1) / (word_counts[word1] + vocabulary_size)  # smoothing
        transition_probs[(word1, word2)] = prob
    return transition_probs
```

Then I chose to plot the data for the top 1000 bigrams on a horizontal bar chart, with the bigrams on the y-axis and the transition probabilities applied to a log scale along the x-axis to be able to present it more neatly.

Explanations of choices:

Laplace smoothing was applied to handle the issue of zero probabilities for unseen bigrams.

In natural language processing, it's common for some word pairs not to appear in a given text, especially in smaller corpora, and though I was only looking at the top 1000 bigrams which would usually not pose any problems with zero probabilities, admittedly the corpora I was running this code through was not particularly large either, so it was used mainly as a safety measure in this case, though this is basically standard in any other bigram calculation use.

Adding on, without smoothing these pairs would have a probability of zero, implying they're impossible, which isn't accurate – it just means they weren't observed.

This becomes problematic when comparing transition probabilities across languages, as missing bigrams can distort the results. Laplace smoothing addresses this by adding one to every bigram count and increasing the denominator by the size of the vocabulary.

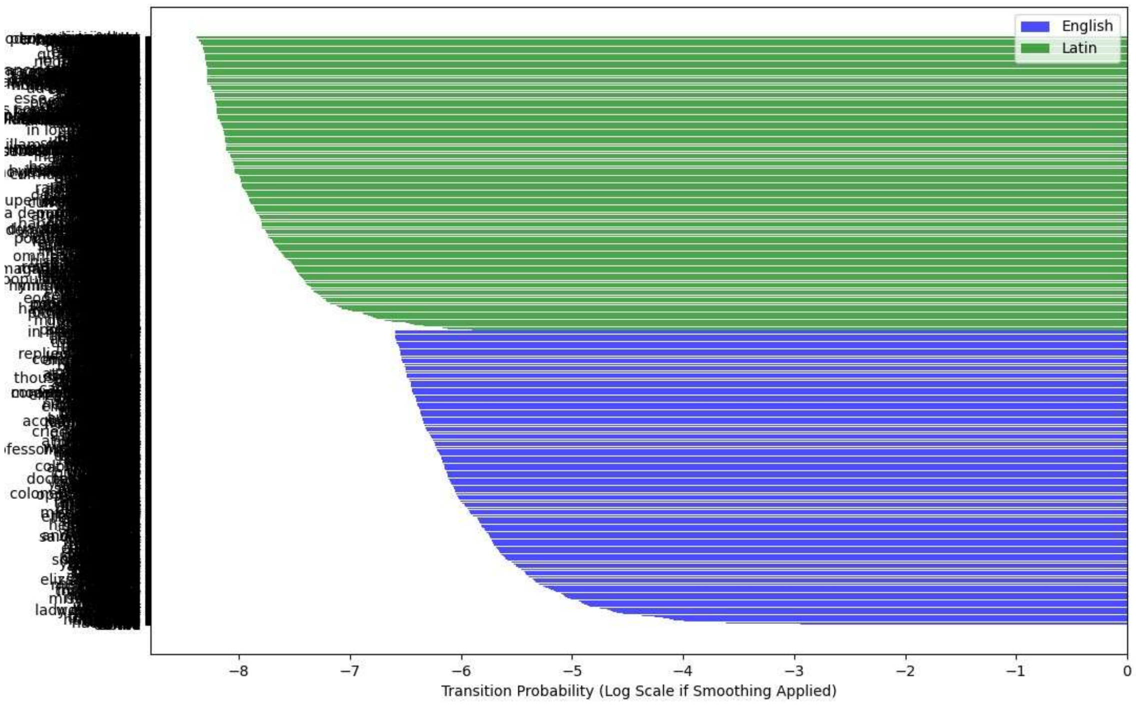
This ensures that even unseen combinations receive a small, non-zero probability.

Overall, the Laplace smoothing approach provides a more stable and fair comparison and improves the clarity of visualizations, especially when plotted on a logarithmic scale, which allows you to compare both larger and smaller probabilities more easily. It also ensured that all probabilities are spread out a bit more evenly and are not too extreme.

Data visualisation explanation:

Graph shown is the bigrams on the y-axis plotted against their probabilities applied on a log scale to be able to read more easily.

For example, if the probability was originally 0.01, after scaling by the natural log scale factor, it would display a value of -4.61 (3 s.f.) and 0.0001 would become -9.21 (3 s.f.). So, as the probability gets smaller, it gets more negative.



Description of graph:

As we can see from the graph, both the English and Latin bigrams displayed a similar exponential shape, though the top 1000 Latin bigrams had smaller probabilities than the English bigrams.

To clarify, the graph shown below displays the top 1000 bigrams in each language, starting with the one with the biggest transition probability at the bottom of the screen.

Results interpreted:

Quite nicely, the observations we can draw from this graph line up with our hypothesis that we would expect to see bigger transition probabilities for the English bigrams than in Latin, as we can see that the highest Latin transition probability starts at around -5.8 whilst the highest English transition probability starts at around -2.9.

Furthermore, the steeper decline in probabilities in the Latin graph suggests that there is a significant drop in the occurrence of bigrams as you move from the most common pairs to the rarer ones, whilst the slower decline in probabilities in the English graph indicates that English has a relatively larger number of moderately frequent bigrams, further reinforcing the hypothesis made at the start.

Therefore, with regards to transition probabilities, the higher transition probabilities in English would be more favourable to an NLP model than Latin.

Final considerations:

Latin is a much lower resource language than English, which means that it would have significantly less data for AI to be trained on.

It must be acknowledged that the tiny amount of data used is by no means representative of either the English or Latin language, it was clearly sufficient in volume (and perhaps accidentally well chosen) such that it was able to support the hypotheses for the two models where hypotheses were made.

Given more time and a more powerful computer, I could have run tests with much larger chunks of training data [and would have most probably obtained the same results].

Conclusion: would NLP models benefit from learning Latin first (based on my data)?

All in all, based on the results coming from my models, we can be thankful that the AI tech boom occurred 1500 years after the fall of the Roman empire, in an Anglophone-dominated world, meaning that NLP models were able to learn natural language more quickly.

Coding Appendix – Feel free to check out the Github account and repository I created, both under the name "TRM-NLP" if you are interested in the full code and specific text files used in the demos.