Kati Provence

Dr. Rashid

CSCI 339

22 September 2025

Development Process, Challenges, Bug Fixes, and Things you Learned:

The first thing I did when developing my portion of the code for this project was to run through a mock example on pen and paper. This helped me better understand what it was that my code needed to do. Initially, deciphering what was already done in the code versus what I needed to do was a challenge. But between reading through the code and comments and my mock run-through on paper, I was able to determine what my tasks were. The preprocessing was already done, as was the math to calculate the log probabilities of each bigram. My task was to go through each token, in this case each letter, and add it to the unigrams and bigrams dictionaries to make the models. Then I would take those probabilities and add them together for each bigram encountered from the test data to make the predictions for both English and Spanish.

One of the first problems I encountered was accessing and saving the values for the bigrams. I first thought that I would combine the previous character and the current one into one string and use *that* as the index for the dictionary. This caused my implementation to fail the test given. While examining the test to determine where the issue occurred, I discovered the correct way to access values in the bigram dictionary. After overcoming this, the largest hurdle to jump was understanding what I needed to do in the `predict()` method, and remembering to add, since we were doing log probabilities.

Do you think it makes sense to create a language model at the character level instead of at the word level for this task? Why?

My initial thought when reading this question was that it would be better to make the model at the word level. This was my instinct because I know that English and Spanish both have the same alphabet. Evidently this is not a problem, as this project was able to train a model at the character level which can successfully determine if a given text is in Spanish or English. I suppose the reason that this works so well is that English and Spanish have some distinct word spelling patterns. For example, many more Spanish words end with a vowel than do English words. These distinct spelling patterns likely contribute to the success of training the model on a character level.

Take a look at the test documents for English and Spanish. Are documents written only in one language?

For the most part, the documents are written in only one language. However, quite a few of the Spanish documents have an introduction written in English specifying things like the title or that it's part of Project Gutenberg. Additionally, one of the pieces of Spanish testing data has an English introduction and conclusion.

What is the minimum number of tokens you need to process to always make the right prediction when testing? You can try with 100 tokens, 200 tokens, etc. You do not need an exact number.

The first thing I did was try limiting the length of the *testing* data, allowing the training data to remain fully utilized. I was interested in how many tokens it took to accurately predict, especially given the fact that some Spanish documents have English intermingled. It took about

1,500 bigrams before the model was able to correctly label most of the documents. It identified all English documents and most Spanish documents correctly. However, there was one Spanish document, the one identified earlier as having a lengthy English introduction and conclusion, that slipped past the detector because it had not yet encountered enough Spanish text. It took about 30,000 bigrams to correctly identify the outlying Spanish document. This number is likely so high because the model had to first get past the English section and then encounter enough Spanish bigrams to outweigh the "English probability" created by the introduction.

Then I tested out how a minimal amount of *training* data affected the model, allowing it to test on the full testing documents. I tested this by limiting the number of bigrams the model was trained on. In contrast to the errors above, where Spanish texts were mistakenly labeled as English, the opposite issue arose when limiting the *training* data rather than the *testing* data. At 500 bigrams in the training model, all Spanish texts were correctly labeled, but only half of the English texts were labeled correctly. It took about 750 bigrams to overcome all inaccuracies.

If you create several models for English and Spanish (using different training data, using different preprocessing, etc.), how can you compare them?

One way to compare the models would be to create visual graphs to compare the different qualities of the models. I would examine number of training bigrams needed for complete accuracy, number of testing bigrams needed for accuracy, flexibility of the model (when encountering imperfect data), and perhaps implementation speed. This would require testing each of the models under the same conditions to discover their differences, advantages, and disadvantages.

Can you train with less training data and still get the right predictions? How does training size affect predictions during testing?

This question was already entertained in part above, "Minimum number of tokens you need to process to always make the right prediction." It is possible to train with less data, however the prediction accuracy does eventually drop when lowering the training data enough. Not only will the training size impact the accuracy of predicting the given texts, but it will also make the system more robust and resistant to variations in the testing data that it hadn't encountered during training. A more extensively trained model will be more likely to accurately predict the language of a text, even if it encounters words or phrases that it hasn't been trained on.

Get some document written in French (e.g., http://www.gutenberg.org/files/36460/36460-0.txt) and use your models to predict the language. How can you justify the predictions?

There were some available French documents in the testing folder, which I made use of to experiment with languages other than English or Spanish. Unsurprisingly, the model labeled a few of the documents were as Spanish. This is likely in part because Spanish and French share articles such as "la." The two documents labeled as Spanish also contained a significant amount of words ending in vowels, which is much more common than Spanish, so it is understandable why they were labeled as Spanish. What was surprising is that the final French document was labeled as English. This may be because there are fewer words ending with vowels. An additional factor contributing to the document being mistaken as English are the French words in the text that were borrowed from English words, or vice versa, such as "parliament".