**CSCE 5290**

**Maia Petee**

### Report - Language Prediction Model

#### Implementation and Bugfixes

After having obtained a working character bigram model that successfully predicts the language of the test documents, I can testify to having learned a lot through bugfixes and progressive implementation. Being relatively new to writing code and completely new to writing complex code, I made sure that I had workable pseudocode and set a number of small goals for myself (e.g. “get working bigram counts” or “get bigram probabilities working”) to help mitigate overwhelm. A number of bugs and inefficiencies cropped up in implementation; many of these have to do with dictionaries, accessing and manipulating dictionaries, and working with several diverse data types in one function. A minor bug involved the probabilities; since I was implementing the float() function incorrectly (taking the float of a number that had already been divided, thereby giving either 0.0 or 1.0), I took some unnecessary time incorporating a Decimal class that forced all numbers to display to six decimal places. Fixing this, and making the entire Decimal class superfluous, was a matter of moving one parenthesis.

The pseudo-code for the prediction step (feeding in the model\_en and model\_es) was not immediately apparent to me, but after help it was much easier. I would say that the most challenging aspect of this implementation was efficiency: when creating the probabilities dictionary, I saved the keys as tuples that included both character and count, and values as log probabilities. The tuple was both extremely inefficient and a roadblock for my final probability counts; I had to create and manipulate a couple of for loops to navigate “around” it and access the probability value I wanted. As such, my code was running, but at an initial time of 6 minutes 30 seconds. By changing the for loops to list comprehensions, I gained 3 minutes, but this was still unacceptable. Eventually, I realized that I could solve the tuple issue entirely by recreating it as its first value: the character (there was no point to the tuples’ existence in the first place; they were just an artefact of my imperfect dictionary creation). This made grabbing and adding the correct probability values a breeze, and my implementation went from 3 minutes to 20 seconds immediately.

#### Results

For this task, I believe that a character bigram model is ideal. Languages have highly individual *phonotactics,*or the sequence(s) and context(s) in which specific characters can occur. Even between two languages that share a character set, there will likely be very telling character distributions (e.g. Language A never allows word-final “e,” but Language B frequently does). The languages don’t have to be entirely dissimilar for these differences to be significant; even a couple phonotactic differences can be sufficient for a correct language prediction. This is the case even though some (a small fraction) of the Spanish training documents is actually in English: the Project Gutenberg header introduces each text in English. The amount of Spanish in these far outweighs in the English, and thus predictions can still be made accurately. In fact, one can predict accurately on a text of as little as two characters. When tested on a document containing *one article* in either English or Spanish, the model was able to guess correctly consistently.

Models can be compared either intrinsically or extrinsically. If comparing intrinsically, one would evaluate the quality of the model on its own: does it represent the language accurately and thoroughly? There is also the issue of perplexity; an ideal model will straddle the line between being needlessly specific and therefore having less data to evaluate with (because it looks for data that does not occur that frequently even in a large sampling of a language, such as a specific 4-gram or 5-gram), or not being specific enough and losing out on predictive power that could have been harnessed. Extrinsically evaluating a model, on the other hand, does so solely on the basis of the benefit to the final application. This approach can be flawed, in that a significantly better language model (which is intrinsically worthwhile) might not translate to any immediate real-world benefit.

When exposed to a French-language document, and only having models for Spanish and English, the model predicted “English.” From a historical viewpoint, this makes sense: the Norman invasion of the 11th century left an indelible French influence on the English language. Many words that were used by educated gentry at the time, and were later incorporated into Modern English, have French origin, as speaking French was highly desirable and a mark of nobility. Thus, many of Modern English’s longer words (“contentment,” “pleasure,”) are barely changed from their original French forms.

Finally, the chart below shows the trend of logarithmic probabilities of one bigram (words starting with “a”) for each language model when the length of the training data was changed substantively. Relative stability and accuracy were maintained until the 2000-word mark, upon which the model started predicting Spanish for everything. This shows that the amount of training data is fairly resilient when it comes to a task as simple as a bigram character model, but that tasks that demand more nuance need a significant amount of training data.

