Part 1:

a. For this part, I considered a token to be every letter, space, and apostrophe. Because I knew that we were using 3 languages with heavy apostrophe use. I also knew that all three languages have different plural forms (I don't actually know that much about French but I don't think they use Ss) so the spaces, to see the ('s', ' ') and ('i', ' ') bigrams a lot would be nice. To preprocess, I removed punctuation (except apostrophes), punctuation, and whitespace.
b. For this part, I decided to use add-one smoothing, so I started every 'gram' at 1 and then added the occurrences of it if it was there. For their respective bigrams, I chose to start it at the number of possible characters. Then, if the bigram exists, I add the number of bigrams to that number.
c. This can't be done without smoothing because, for example, comparing the English ('\'', 's') to anything in Italian will force us to divide by 0. Add-one smoothing worked well enough for this assignment because the number of unseen events was not extremely high, and because only using bigrams makes the likelihood of unseen events lower.
d. The results from my program were right 297/300 times, for an accuracy of 99%.

Part 2:

a. For this task I considered a word to be anything separated by space, and apostrophes. Ideally, I would have the clitic and the apostrophe attached as a together, such as '\'s' or 'j\'', but that's not what I wound up doing. Again, for this one, I excluded all numbers and punctuation aside from apostrophes, and removed all whitespace.
b. For out of vocabulary items, I did the same thing as the letter program. I added one to every unigram and the size of the corpus to every bigram, and then added their respective counts if they existed.
c. Again, this model needs smoothing because we're comparing to languages that might not have the same words in them.
d. My program was right 294/300 times, for a result of 98%. This could have been better if I had included sentence end and beginning tokens instead of cleaning them.

Part 3:

a. This, for the same reasons, has to be done with smoothing. Because the number of items seen once might be unreliable, we can use the multiple frequencies above 0 as long as they exist. For an unseen word, we can consider every frequency above it until there is one that has a non-zero frequency. This gives us a more reliable amount of low-frequency items. To smooth unseen words, we lump them in with the low-frequency events, which all get smoothed, where we consider the count from the above frequency, and divide it by the total number of 'events', multiplied by the frequency of their respective frequencies.

Observations

The good-turing model was the best by far. However, I think that if my cleaning were as thorough and tuned on the word level as it was on the letter level, that would be better as well. As is, the letter model comes in second place.