Ryan Voldstad
Mario Villaplana
CS276 PA 2 Report

Model Building

models.py builds numerous models to aid in the spelling correction process. First, we build data structures to help the language model. The LanguageModel class first builds counts of the unigrams and bigrams in the corpus by iterating through the files in the corpus one word at a time. Along the way, we build an inverted index mapping from bigrams to words containing that bigram whose use for candidate generation will be described later. After building these data structures, we normalize the counts to produce unigram and bigram probabilities, along the way building a dictionary mapping from word length to words of that word length, another data structure used for candidate generation. We then save the unigram and bigram probabilities as well as the bigram inverted index and length dictionary to disk.

The EditModel iterates over all the pairs in edit1s and finds the single edit between the two words. While finding this edit, it also adds bigram and unigram (character) counts from the correctly-spelled words to dictionaries. Once the edit has been found, it is added to the appropriate 2-dimensional dictionary (modeling a matrix, we don't store the 0 values for efficiency), keyed by the two appropriate characters involved in the edit. After iterating through all pairs, edit probabilities are computed in log space- using edit counts, unigram/bigram counts, and Laplace smoothing- and then written to disk. The character bigram and unigram counts are also written to disk for use (on unseen edits) in the corrector.

Spell Corrector

For performing the actual spell correction, we first read in the query and model data off of disk. Then, when calculating spell corrections with the uniform edit cost option, we iterate through each misspelled word in the query, performing the following operations:

- If the word isn't in the dictionary, generate candidates for replacing the word by finding words in the dictionary whose lengths are no more than 2 less or 2 more than the length of the original word.
- Compute the Jaccard coefficient of each of these words with the original word. If it's less than 0.5 and the word is within edit distance 2 of the original word, add this word as a candidate.

When calculating spell corrections with the empirical edit cost option, we do the same as above except we also calculate the empirical edit probability. This is calculated using a modified edit distance algorithm which uses the dictionaries built in the edit model for each edit cost. For all character pairs not found in the corresponding dictionary, Laplace smoothing is used.

Additionally, for each match, we apply a small penalty under the assumption that the probability of making any single edit is .05.

After generating candidates, we calculate the score for a candidate by computing $\log(P(Q) * P(R|Q)) = \log(P(Q)) + \log(P(R|Q))$, adding the best candidate to the corrected query. For $P(Q)$, we use the language model and use interpolation with a weight of 0.2 for unigram probabilities and 0.8 for bigram probabilities. When using uniform edit distance, we give each edit a probability of 0.1. When using empirical edit distance, we use the empirical edit probabilities that were already calculated as described above.