1.Models.py

We iterated through corpus word by word and created two dictionaries (one for unigram count and the other for bigram count). We also kept track of total term_count and at the end, wrote on the disk using serialize_data

For analyzing edit1, we wrote a function to find the difference (edit) given two queries. Using the helper function, we logged all the edits made in the queries, and additionally we kept track of all the counts of each letter and bi-letter. Again, we wrote the learned data on the disk, so that it can be called easily by the correcter.py

2. Candidate Generation

First, we considered an input query as one long string and we generated all 1-edit candidates by doing 1 operation on the query. Then, we filtered out queries that contained any non-dictionary word. With this set of candidates, we were able to achieve 77% accuracy (compared to gold.txt in uniform cost editing mode).

From this result, we found that many of the incorrect guesses were due to a low number of 1-edit distance candidate set. So, we decided to generate more candidates by also looking at the 2-edit distance candidates. However, because it would be too expensive to generate 2-edit distance candidates for all the input queries, we only looked at them for queries with low number of 1-edit distance candidate set.

From the 1-edit distance candidates, we kept queries without any non-dictionary word and also filtered out queries with only one non-dictionary word. For input queries with less than 10 1-edit candidates, we generated another set of 1-edit distance candidates from previously filtered out 1-edit, 1 non-dictionary candidates, creating 2-edit distance candidates (again, we only used 2-edit candidates without any non-dictionary terms). With this increased set of candidates, we were able to achieve 82% accuracy on gold.txt in uniform cost editing mode.

-- we found out that in generating 1-edit distance queries, we left out cases where there is deletion (from candidate letter to input query) at the very end of the string. However, we intentionally left this part out because testing with the fix shows no improvement and slows down the program significantly.

3. Scoring

(All probabilities are calculated in log space)

P(Q) is calculated by the learned language model. if Q = q1q2q3..., P(Q) = P(q1)P(q2|q1)P(q3|q2)... and the bigram probability is calculated using the N-gram interpolation with lambda = 0.2

P(R|Q) for uniform cost edit model is calculated considering Q as one large string. We set the edit probability of 0.01 and if len(Q) = q and edit distance = e, P(R|Q) = 0.01^e * (1-0.01)^(q-e)

For the empirical model, we keep track of edits made to each candidate query, and we return the edit probability accordingly using the channel model described in the lecture slide (with add-1 smoothing). To give more weight to edits that hasn't been seen in our dev set, we assigned even lower probability for a candidate with such edits.