System design:

The system's design is close to the skeleton code. The edit cost model interface is takes in a list of edits instead of the edit distance as the edit distance, inferred by the number of edits, is used in uniform cost model and the edits is used in the empirical edit model.

I have created a bunch of other utility classes to:

- Find edit distance
- Find the edits between two strings
- A class to hold a word and its possible candidates (limited to 2 edits)
- Calculate the probability of a candidate query
- K-gram indexing and retrieval

Methods used:

I used dynamic programming to find the edit distance between two strings. I optimized it to use O(n) memory since only two rows are needed to calculate the edit distance. Using a similar approach, but with no optimization, I listed out the edits between two given strings by backtracking the matrix.

Smoothing is performed as suggested in the spec. Bigram probabilities are interpolated with unigram probabilities. Candidates are scored with different weights. Laplace add-one smoothing is used in calculating edit probabilities in the empirical model.

Candidate Generation:

I first find out the possible candidates for each word in the given query. This is done by finding the edit distance between each word in the query with each word in the dictionary. Then I find out the edits between the words only if the edit distance is <=2 as they constitute 97% of spelling errors.

Then I find out the possible candidate queries by as follows:

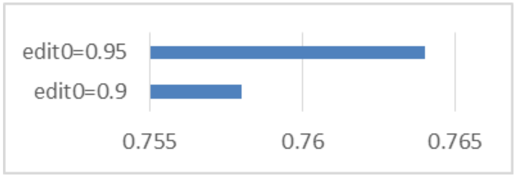
- One edit candidate queries:
    - Cross product of 1 edits possibilities (includes splitting a word into two and joining adjacent words) of each word in the query with the rest of the original words in the query.
- Two edit candidate queries:
    - Cross product of 2 edits possibilities of each word in the query and the rest of the original words in the query.
    - Cross product of 1 edit possibilities of each word in all possible pairs of word in the query with the rest of the original words in the query.

I check whether the rest of the original words in the query are valid dictionary words before taking the cross product to save time.

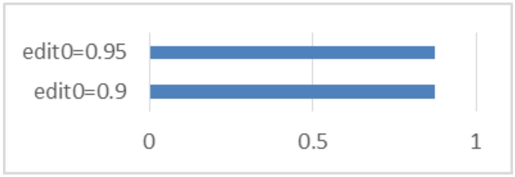
Parameter Tuning:

Here are the various performance data collected on tuning various parameters:

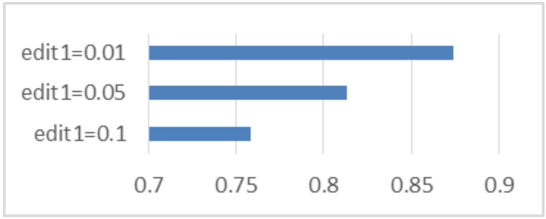
Uniform	lambda=0.1	mew=1	edit1=0.1
edit0	accuracy	time	
0.9	0.758	765	
0.95	0.764	768	



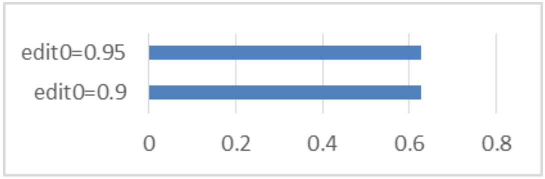
Uniform	lambda=0.1	mew=1	edit1=0.01
edit0	accuracy	time	
0.9	0.874	777	
0.95	0.874	780	



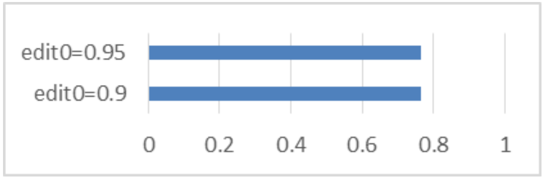
Uniform	lambda=0.1	mew=1	edit0=0.9
edit1	accuracy	time	
0.1	0.758	765	
0.05	0.813	784	
0.01	0.874	777	



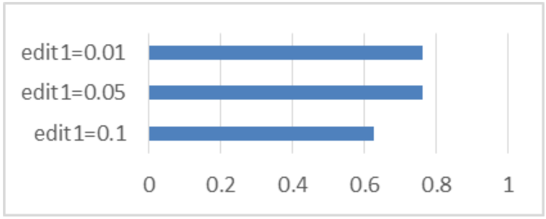
Uniform	lambda=0.1	mew=2	edit1=0.1
edit0	accuracy	time	
0.9	0.628	779	
0.95	0.628	762	



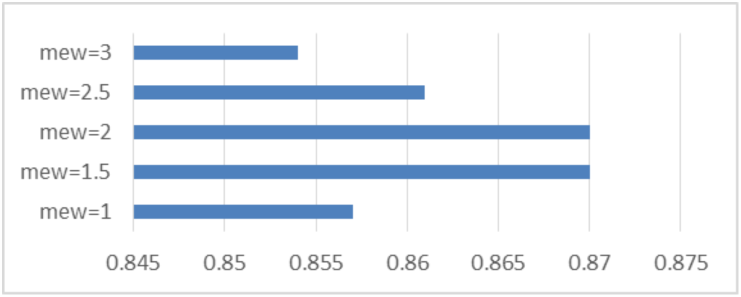
Uniform	lambda=0.1	mew=2	edit1=0.01
edit0	accuracy	time	
0.9	0.764	771	
0.95	0.764	769	



Uniform	lambda=0.1	mew=2	edit0=0.9
edit1	accuracy	time	
0.1	0.628	779	
0.05	0.764	771	
0.01	0.764	763	



Empirical	lambda=0.1	
mew	accuracy	time
1	0.857	779
1.5	0.87	785
2	0.87	773
2.5	0.861	794
3	0.854	807



K-Gram	mew=2			
Low	High	Split	Accuracy	Time
0.2	0.45	8	0.848	714
0.2	0.4	6	0.857	573
0.3	NA	NA	0.859	686

Using K-Gram gives a boost of 200 seconds for a tradeoff of 1.5% accuracy.