Programming Assignment 2

Tayyab Bin Tariq, Rukmani Ravi

Language Model

For the purpose of ranking a string of words as being generated from a language similar to the one that generated our corpus we trained a bigram language model. This model was created by counting the occurrence of all vocabulary terms and bigrams. The bigram model is then smoothed with Laplacian smoothing for k=1. The unigram model does not need smoothing because of the closed dictionary assumption. We combined these two models with N-gram interpolation. As advised, we started with the interpolation parameter lambda = 0.2. After implementing the rest of the spell corrector, we tuned tested our code with lambda = 0.1, 0.2, 0.3 and 0.4. The performance of these parameters is shown in the table below with all other parameters held fixed.

Lambda	**Accuracy (Uniform)**	**Accuracy (Empirical)**
0.1	82.55%	84.70%
0.2	82.16%	84.31%
0.3	81.5%	83.92%
0.4	81.18%	83.53%

Extra Credit Extension

In addition to using n-gram interpolation we also used stupid backoff with lambda = 0.2. This gave us an accuracy of 81.57%.

Noisy Channel Model

Uniform Cost Model

For the uniform cost model, we used a uniform probability of p=0.05 for errors that were one edit distance from the candidate correction and a probability of p^2 = 0.0025 for errors two edit distance away from the candidate. A value of 0.95 is used to denote the entered word being the correct one given it exists in the dictionary. The following table shows the accuracy values for different p values. Clearly most of the time, what the user entered is correct. Only 1% of the time, we need a spelling correction in this dataset.

P	**Accuracy (Uniform)**
0.01	86.27%

0.05	82.55%
0.10	76.47%

Empirical Cost Edit Distance

For the empirical cost edit distance we calculated the probability of insertions, deletions, substitutions and transpositions as stated in the handout. This was accomplished by computing the edits between the entered word and the candidate. Each of type of the edit was counted together with its context and normalized as suggested in the handout. This model is then used at the time of candidate generation to assign a probability of a particular candidate based on the number and types of edits it has. The edits are assumed to be independent and if a suggestion has more than one edit the probabilities of the edits are simply multiplied. This is done to discount more than edits more heavily. The model is also smoothed with Laplace smoothing for k=1. The following table shows different accuracy values for change in values of lambda, Mu and P. Where P is the probability that an entered query is the intended query.

Mu	**Lambda**	**P**	**Accuracy (Empirical)**
1	0.1	0.95	84.71%
2	0.1	0.95	83.73%
3	0.1	0.95	82.16%
1	0.2	0.95	84.31%
1	0.3	0.95	83.92%
1	0.4	0.95	83.53%
1	0.1	0.99	84.90%
1	0.1	0.95	84.71%
1	0.1	0.90	84.51%

Candidate Generation

We based our code on Peter Norvig’s blog post about spelling correction. In addition we also considered the errors where two words were joined by a space by incorporating the space character in our alphabet set. We also considered the cases where two words have been separated by a space at the time of input. To reduce the space of generated candidates we use the probability distribution of edits in our edits training corpus. We filter out all candidates that are generated by edits that occur in our training corpus with probability smaller than a tuned constant parameter. This greatly reduced the size of the candidate set.

Candidate Scoring

The candidates are scored based on the formula given in the assignment statement. At first we used mu=1. We also tested with mu=1, 2 and 3. The following table shows the accuracy of our system for different values of mu.

Mu	**Accuracy (Uniform)**	**Accuracy (Empirical)**
1	82.55%	84.71%
2	82.35%	83.73%
3	81.57%	82.16%

Extra Credit

For extra credit we implemented stupid back off as part of our language model. We used different values of lambda parameter used when backing off to the unigram probability for the stupid backoff model. The following table shows accuracy for different values of lambda. We can conclude that an interpolated language model seems better in this case than simply backing off to a unigram model.

Lambda	**Accuracy (empirical)**	**Accuracy (uniform)**
0.2	81.57%	76.47%
0.4	78.6%	73.52%
0.6	76.47%	70.78%

We also tried to implement an n-gram inverted index for candidate generation however, it did not prove to be very useful and cause the accuracy of our system to drop to about 60%. We therefore dropped the idea and implemented stupid backoff instead.