CS276: Information Retrieval and Web Search **Spring 2013**

Programming Assignment 2

Botao Hu (botaohu), Jiayuan Ma (jiayuanm) Completed Date: *May 4, 2013*

1 Language Model

We build our language model using the provided corpus, and count all the unigrams and bigrams that appeared in the corpus. Using these counts, we can estimate the parameters in our language model by calculating

$$P_{mle}(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_i)} \qquad P_{mle}(w_i) = \frac{count(w_i)}{\text{total number of terms}} \tag{1}$$

To deal with the data sparsity problem, we use N-gram interpolation to obtain the final conditional probability in our language model.

$$P_{int}(w2|w1) = \lambda P_{mle}(w2) + (1-\lambda)P_{mle}(w2|w1) \tag{2}$$

In our experiment, we use the default $\lambda = 0.2$. In addition, we use all the unigram in the corpus to build a bigram-character index to support efficient candidate word generation.

2 Candidate Word Generation

We use bigram index to efficiently compute Jaccard distance between words, threshold Jaccard distances to obtain a list of candidates, and filter the candidates using the exact Damerau-Levenshtein distance. We tune the threshold parameter to be 0.5 to balance between efficiency and accuracy.

3 Candidate Query Scoring Using Markov-Chain Inference

For every term in a given query, we can

- split it into two words in the dictionary
- edit it using at most two edit operations (deletion/insertion/substitution/transposition)
- combine it with the next word to form a new word that appeared in the dictionary

Given the proposed candidate query, we compute our language model using

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2) \cdot P(w_n|w_{n-1}) \tag{3}$$

which is essentially a Markov-Chain model. We can minimize the negative log-likelihood

$$-\log P(w_1, w_2, \ldots, w_n) = -\log P(w_1) - \sum_{i=2}^{n} \log P(w_i|w_{i-1}) \tag{4}$$

using dynamic programming (Viterbi algorithm) on markov chain. Based on this approach, we can find a global minimum without scoring all the possible candidate queries.

4 Edit Cost Model

Based on the above discussion, we can easily model edit cost of each word using singleton potential in our Markov chain model. The edit cost is given as follows.

$$P(x|w) = \begin{cases} \frac{del[w_{i-1},w_i]}{count[w_{i-1},w_i]} & \text{if deletion} \\ \frac{ins[w_{i-1},w_i]}{count[w_{i-1}]} & \text{if insertion} \\ \frac{sub[x_i,w_i]}{count[w_i]} & \text{if substitution} \\ \frac{trans[w_i,w_{i+1}]}{count[w_i,w_{i+1}]} & \text{if transposition} \end{cases} \tag{5}$$

All these count statistics come from the provided training file. For the unseen edits, we use Laplace smoothing to deal with the data sparsity issue.

5 Levenshtein Automata

We observe that Jaccard distance is not a good approximation of Damerau-Levenshtein distance, especially when the words are short and the correct edits are transpositions. To fix this problem, we implement the Levenshtein Automata to support *fast* and *accurate* candidate generation for words.

6 Conclusion

Our ultimate model (Empirical Edit Cost, Levenshtein Automata with Dynamic Programming) achieves 89.45% accuracy on the provided development queries.