CS 276 Programming Assignment 2 Project Report

Jiaji Hu, Xuening Liu

1 System Design

1.1 Code Structure

The following is a description for the main classes for this assignment:

`BuildModels` and `RunCorrector` are used to build models and run the spell corrector, respectively.

`LanguageModel` and `NoisyChannelModel` builds the language and noisy channel models during model building time, which are saved to disk and loaded when needed in the future.

`EditCostModel` is the interface, while `UniformCostModel` and `EmpiricalCostModel` are the cost models used to compute edit probabilities.

1.2 Model Building

1.2.1 Language Model

While building the language model, we go through the training corpus, building up our dictionaries. We document the unigram and bigram frequency to be used for calculating probabilities at query time. For our alternative smoothing methods (in particular Kneser-Ney), for each word, we also need to document the number of unique bigram continuations and number of possible following words.

To conform with memory requirements and still save the information for the extra credit, we implemented a `Triple` class and a `TriDictionary` class to store `<String,Triple<Integer,Integer,Integer>>` entries.

1.2.2 Noisy Channel Model

To build the noisy channel model, we go through the training edit file, which is used to train the empirical cost model.

********** STUFF HERE **********

1.3 Error Correction

At query time, the input query is fed into the candidate generator to generate candidates with edit distance lesser or equal to 2 (including the original query at edit distance 0). The candidate generation process is discussed in detail in Section 3.

After candidates are generated for a given query, they are assigned probability scores. To choose the best candidate, we want to calculate $P(Q|R)$, where R is the actual input, and Q is the candidate. We know $P(Q|R) \propto P(Q)P(R|Q)$, where $P(Q) = P(w_1, w_2, \ldots, w_n)$ is calculated with the language model, and $P(R|Q)$ is calculated using the noisy channel model. For scoring, we use $P(Q|R) \propto P(Q)^{\mu}P(R|Q)$, where μ is a parameter we can tune for performance.

For the language model, $P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w_{n-1})$. We use maximum likelihood estimation for these unigram and bigram probabilities, with smoothing for the bigram probabilities to account for unseen bigrams. Smoothing methods will be discussed in depth in the next section.

For the noisy channel model, if we are using uniform edit probabilities, then we simply assume that any edit of edit distance one occurs with the same small probability (e.g. 0.01), therefore, changes of edit distance 2 will have 0.0001 probability. On the other hand, if we are using the empirical cost model,

********** STUFF HERE **********

2 Methods

2.1 Smoothing for Edit Probabilities

2.2 Smoothing for Language Model

2.2.1 Original method (Jelinek-Mercer)

Smoothing is done to try to account for the data sparsity problem, where there may be a large amount of bigrams that never appeared in the training corpus. If a query had such an unseen bigram, we don't want its probability to be zero.

The assignment instructions suggest we interpolate unigram probabilities with bigram probabilities to get the final conditional probability like so:

$$P_{int}(w_2|w_1) = \lambda P_{mle}(w_2) + (1-\lambda)P_{mle}(w_2|w_1) \tag{1}$$

In practice this smoothing method gives good results. However, we can still explore other possibilities.

2.2.2 Katz (back-off) method

When trying to estimate $P(w_2|w_1)$, the backoff model[2] uses $P_{mle}(w_2|w_1)$ if $count < w_2, w_1 >$ is larger than some threshold (e.g. 0). Else, the model backs off to use $P_{mle}(w_2)$.

In doing this, the backoff model tries to use the most reliable probability to estimate the conditional probability. However, it takes testing to decide on the back-off threshold, as well as whether to weight the probabilities.

In practice, we tried using threshold = 0, 1, 2, and weighting $P_{mle}(w_2)$ with λ. However, the performance of the system changed for the worse, compared to the interpolation smoothing method. From our readings, it seems that the Katz smoothing model works best for large training sets.

2.2.3 Absolute Discounting and Kneser-Ney Smoothing

Like Jelinek-Mercer, Absolute Discounting uses the interpolation of a higher-order and lower-order models. What is different is that instead of multiplying the higher order probability with a weight, we subtract a fixed amount d from the count (the nominator).

Looking at the Good-Turing numbers from [1], we find that count c and Good-Turing c* seem to have a correlation of $c^* \approx c - 0.75$. Therefore it may be a good idea to start with 0.75 as d.

Kneser-Ney smoothing extends absolute discounting interpolation with the following idea: The lower-order model is only significant when the high-order model gives a small value (the count is small or zero). Therefore, the lower-order model should be optimized so that it gives low probabilities for words that usually appear as the second word in a combination. (For example, the lower-order model should give a low score for the word "Francisco".)

To do this, we compute the lower order model with the nominator being the number of unique bigram continuations made by the word, instead of total bigrams with that word. In this case, if "Francisco" appears as the second word in a bigram 50 times as "San Francisco" and once as "Mr Francisco", its nominator will be 2 instead of 51, effectively capturing the fact that the word appears heavily in one unique bigram and lowering its probability score.

The equations are as follows:

$$P_{KN}(w_i|w_{i-1}) = \frac{max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{CONT}(w_i) \tag{2}$$

Where

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}| \tag{3}$$

$$P_{CONT}(w) = \frac{|\{w_{i-1:c(w_{i-1},w)>0}\}|}{\sum_{w'}|\{w' : c(w'_{i-1}, w') > 0\}|} \tag{4}$$

After we implemented Kneser-Ney smoothing, we used the new smoothing method for our corrector. After tuning parameters such as μ, we were able to get 90.7% accuracy using just uniform edit costs, a slight improvement on basic interpolation smoothing.

3 Candidate Generation

4 Parameter Tuning

Algorithm	index time(s)	index size (MB)	index size - including dicts (MB)	average retrieval time (s)
Basic	67	58	72	1
VB	149	18	33	1
Gamma	178	13	29	1

References

[1] Kenneth W Church and William A Gale. A comparison of the enhanced good-turing and deleted estimation methods for estimating probabilities of english bigrams. *Computer Speech & Language*, 5(1):19–54, 1991.

[2] Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.