CS 276 Programming Assignment 2 Project Report

Jiaji Hu, Xuening Liu

1 System Design

1.1 Code Structure

`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

To building the language model, we go through the training corpus, building up our unigram and bigram dictionaries. For alternative smoothing methods, 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 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. The candidate generation process is discussed in detail in Section 3.

After this, 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 $P(Q)$ 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 tune for performance.

For the language model, $P(Q) = P(w_1, \ldots, w_n) = P(w_1)P(w_2|w_1)\ldots P(w_n|w_{n-1})$. We use maximum likelihood estimation for these unigram and bigram probabilities, with smoothing for the bigram probabilities. Smoothing methods are 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. If we are using the empirical cost model,

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

2 Methods

2.1 Smoothing for Edit Probabilities

To assign probabilities for edits that are not found in the training edit file, we first used simple add-one smoothing. To try to improve on that, we switched to add-delta smoothing, and tested some values of delta.

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

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. If a query had a bigram that was not seen during training, 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:

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

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 $P_{mle}(w_2)$. This also works with weights. In doing this, the backoff model tries to use the most reliable probability to estimate the conditional probability.

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

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.

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 idea that the lower-order model is only significant when the high-order model gives a small value (the count is small or zero). Therefore, the it should be optimized to give 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. The equations are as follows:

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

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

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

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

Candidate generation is very important. On one hand, if the right correction was not even generated, the output query cannot be correct, so we want to cover as much space as possible. On the other, generating and scoring large amounts of candidates puts a huge strain on runtime and memory, so there is an obvious trade-off here.

With this tradeoff in mind, we looked at three strategies for generating distance 2 candidates, in the order of increasing number of candidates generated: (note that we define a candidate to be "valid" if all its words appear in the unigram dictionary.)

1. Only generate distance 2 candidates if there is no valid distance 1 candidate.
2. Generate distance 2 candidates from all *valid* distance 1 candidates.
3. Generate *all* distance 2 candidates.

4 Parameter Tuning

Algorithm	index time(s)	index size (MB)	index size - includin
Basic	67	58	72
VB	149	18	33
Gamma	178	13	29

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.