CS276 PA2 Report

Jiawei Yao
jwyao@stanford.edu

Wei Wei
wwei2@stanford.edu

May 1, 2014

1 System Design

When designing the system, we put an emphasis on the following aspects:

- **Modularity**: conform to good OOP practices; favor abstraction over implementation; reduce coupling among different components.
- **Flexibility**: it should be easy to add/remove implementation of a module without breaking other parts of the system.
- **Efficiency**: the system should be as performant as possible.

An example of modularity is we want to use `LanguageModel` as source of vocabulary/lexicon for `CandidateGenerator`, but we don't want to couple `CandidateGenerator` with the concrete `LanguageModel`. So we introduced an interface `Vocabulary` and let `LanguageModel` implement that interface.

Examples of flexibility include that we want to implement various smoothing techniques on language model, so we make `LanguageModel` an abstract base class and let subclasses to implement `bigramProbability`. With this change, we are able to add different smoothing techniques with few code.

The system is modularized into three parts:

1. **Language Model**, used to compute the $P(Q)$ part of the noisy channel model.
2. **Noisy Channel Model**, used to compute the $P(R|Q)$ part.
3. **Candidate Generator**, used to generate possible candidates of Q given R.

Each part is documented in detail in the following sections, with design decisions addressing the above aspects.

2 Language Model

We referred to [4] and [1] for advanced smoothing techniques. There are 3 language models with different smoothing techniques in our implementation:

1. **InterpolationLM**: implements the linear interpolation described in PA2 description.
2. **AbsoluteDiscountLM**: implements absolute discounting described in [1].
3. **KneserNeyLM**: implements Kneser-Ney smoothing described in [1].

Specifically, since Kneser-Ney smoothing is an extension of absolute discounting, we implemented `KneserNeyLM` as a subclass of `AbsoluteDiscountLM`, which demonstrates clear modularization and extensibility of the system.

A specific optimization we found worthwhile when building the language model is to avoid compute all possible bigram probabilities. If all possible probabilities are computed, the memory limit will be exceeded, because the vocabulary size is huge. Instead, we store raw $count(w)$ and $count(w_{i-1}w_i)$ and compute the probability with the counts on the fly when needed. Storing counts also has the advantage that they can be used by advanced smoothing techniques to generate higher-order data, such as n_1 and n_2 used in absolute discounting and $N_+(\cdot w_i)$ in Kneser-Ney smoothing.

On the other side, precomputing information of 1-dimension, higher-order data can greatly improve performance. For example, computing $N_+(\cdot w_i)$ requires traversing all bigram counts. We can traverse the counts once and cache the result, or we can traverse the counts every time we need to compute $N_+(\cdot w_i)$. Clearly, the former approach would be much faster as $|(w_{i-1}, w_i) : count(w_{i-1}w_i) > 0|$ is large. This is an example of trading space for time.

3 Noisy Channel Model

Edit[3][2]

4 Candidate Genenration

Viterbi...

5 Parameter Tuning

Graphs...

6 Extra Credit

Yes, please!

References

[1] Chen, Stanley F and Goodman, Joshua. *An empirical study of smoothing techniques for language modeling*. Proceedings of the 34th annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, 1996.

[2] Kernighan, Mark D., Kenneth W. Church, and William A. Gale. *A spelling correction program based on a noisy channel model*. Proceedings of the 13th conference on Computational linguistics-Volume 2. Association for Computational Linguistics, 1990.

[3] Jurafsky, Dan, and James H. Martin. *Speech & Language Processing*. Pearson Education India, 2000, pp. 163–168

[4] Jurafsky, Dan. *Language Modeling*. 2014, available at `http://www.stanford.edu/class/cs124/lec/languagemodeling.pdf`