CS 388 Natural Language Processing Homework 1: N-gram Language Models

Due: Feb. 18, 2015

In this assignment you will explore a modification of a simple, typical N-gram language model and experimentally test it on English data.

Existing Bigram Model

Java code for a simple bigram language model is on the department file server in `/u/mooney/cs388-code/nlp/lm/BigramModel.java`. The code uses simple linear interpolation with a unigram model for smoothing, just using fixed weights (0.9 and 0.1) for combining the bigram and unigram model. During training, it replaces the first occurrence of a token with < UNK> for "unknown" in order to handle novel (out-of-vocabulary, OOV) words encountered in testing (which are also replaced with < UNK>). It computes complete sentence probability and perplexity in log space in order to avoid problems with floating-point underflow when multiplying many small probabilities. Read the comments in the code to familiarize yourself with this short program.

This model can be trained and tested on sentence-segmented data in the Linguistic Data Consortium (LDC)'s Penn Treebank collection. See the part-of-speech tagged (POS) data in `/projects/nlp/penn-treebank3/tagged/pos/`. The class `POSTaggedFile` in `/u/mooney/cs388-code/nlp/lm/` contains code for extracting tokenized, sentence-segmented text from these POS-tagged files, see the comments in this file for details.

The `main` method for `BigramModel` trains and tests the model on LDC tagged files. It takes a list of tagged files and/or directories as arguments and its last argument (a real value between 0 and 1) is the fraction of the sentences in this data to use for testing. See a trace file running this program. It shows a trace of training and testing the model on the `atis` (airline booking query), `wsj` (Wall Street Journal text) and `brown` (Brown corpus of mixed-genre text) corpora using the first 90% of the sentences for training and the last 10% for testing. It shows perplexity for both the training and held-out test data. The additional "Word Perplexity" measure excludes the prediction for the end-of-sentence (</S>) token, which will be used for the comparisons discussed below.

Your Task: Backward and Bidirectional Bigram Models

Standard N-gram language models model the generation of text from left to right. However, in some cases, tokens might be better predicted from their right context rather than their left context.

Your first task is to produce a "backward" bigram model that models the generation of a sentence from right to left. Think of a very simple way to very quickly use `BigramModel` to produce a `BackwardBigramModel` that is identical except for the modeling direction. The one significant difference between a left-to-right and a right-to-left (backward) model is that the backward model's final prediction is to predict sentence-start (< S>) rather than sentence-end (</S>). Therefore, the "Word Perplexity" version of the evaluation metric that ignores the prediction of sentence boundaries in either direction, is the fairest way to compare these two models.

Test your backward model on the `atis`, `wsj`, and `brown` corpora and produce a trace analogous to the `BigramModel` trace file for `BackwardBigramModel`.

Next build a `BidirectionalBigramModel` that combines the forward and backward model. When determining the probability of a word in the sentence, it should use both the estimate from its forward and backward contexts by linearly interpolating the probability predicted for that token by the `BigramModel` and the `BackwardBigramModel`. By default, just weight the prediction of both models equally when interpolating a probability for each word/token in the sentence, although you may try different weights if you believe the comparative results from the forward and backward model warrant that they be weighted differently.

Test your bidirectional model on the `atis`, `wsj`, and `brown` corpora and produce a trace analogous to the `BigramModel` trace file for `BidirectionalBigramModel`.

Both your `BackwardBigramModel` and `BidirectionalBigramModel` should take the same input and produce the same style of output as `BigramModel`, except you only need to show the "Word Perplexity" for the bidirectional model. That is, for the bidirectional model you should not try to combine the probabilities for predicting sentence-start and sentence-end since these are not comparable tasks, so just don't include sentence boundary prediction in the perplexity score at all.

Submission

Using Canvas to submit your code, output trace files, and a well-written, coherent, well-formated project report (about 2 pages) that includes a nice well-structured table of comparative results and includes a discussion that answers at least the following questions:

1. How does the "Word Perplexity" of the backward bigram model (for both training and test data) compare to the normal model? Discuss the reasons for any diferences or similarities found.
2. How does the "Word Perplexity" of the bidirectional model (for both training and test data) compare to both the backward model and the normal model? Discuss the reasons for any diferences or similarities found.