CS 6320 Natural Language Processing

Homework 2 N-gram Language Model

Due: Sep 28th, 2014

I. Written questions [10 points]

Explain in your own words how language models are used in the following tasks.

(a) Machine translation, (b) information retrieval.

II. Programming part [90 points]

You will implement a bigram language model with Katz backoff smoothing. You will compute the perplexity of the model on the test set and write up a report about your findings. **This assignment will take some time, so plan ahead wisely.**

The training and test data are from some news article text.
We have done some preprocessing of the data (split into sentences, tokenization, cleaning). Please use the data as is for your homework.

Each line in the training and testing files contains a sentence (with the sentence starting <s> and ending symbol </s>).
Use white spaces as separator to build a word-based language model.
Please treat sentence beginning and ending symbols as a "word".

You probably should consider using a language that is efficient in string processing and maybe associate array.

Training set: http://www.hlt.utdallas.edu/~yangl/cs6320/homework/hw2_train

Test set: http://www.hlt.utdallas.edu/~yangl/cs6320/homework/hw2_test

There are two steps in this assignment.

1. Training.

Implement a program that reads in the training file, and outputs a LM file listing the unigrams with their estimated probabilities and backoff weights, and the bigrams with their estimated probabilities.
The file format of your LM should look like this:

unigrams:
probability word backoff_weight
.....

bigrams:
probability w1 w2

where "backoff_weight" in the unigram line means the weight alpha(h) used in backoff (see below) when this word is a history for an unseen bigram,
and the two words in bigram line means w2 follows w1, i.e., the bigram probability is p(w2|w1).

Please use an ascii file for the model such that it is easily readable.

Since later we will use logprob during testing, it's a good idea to store the probabilities and backoff coefficients using log values in the model file.

In this assignment we only ask you to implement a simple Katz backoff model --- use the Good-Turing estimate for bigrams occurring only once, and the maximum likelihood estimate for all other bigrams and unigrams. That means:

(a) if C(h,w) occurs more than once in the training set, $P_{ml}(w|h)=C(h,w)/C(h)$;

(b) for a bigram (w,h) that occurs only once in the training set, the Good-Turning estimate is:
$P_{GT}(w|h)=2*N_2/(N_1*C(h))$ where N_2 and N_1 mean the number of bigram types that occur twice and once respectively;

(c) for a bigram that never occurs in the training set, $P_{katz}(w|h)=\text{alpha}(h) * C(w)/N=\text{alpha}(h) * P_{ml}(w)$, where N is the total number of unigram tokens, and alpha (h) is the backoff for the unigram h:

$$\alpha(h) = \frac{1-\sum_{w:C(h,w)>0} P(w \mid h)}{1-\sum_{w:C(h,w)>0} P_{ML}(w)}$$

$$= \frac{1-\sum_{w:C(h,w)>1} P_{ML}(w \mid h) - \sum_{w:C(h,w)=1} P_{GT}(w \mid h)}{1-\sum_{w:C(h,w)>0} P_{ML}(w)}$$

(plug in case (a) and (b) for the numerator).

Note 1: The bigrams don't cross sentence boundaries, i.e., you don't have bigram probability p(<s>|</s>) (the sentence beginning token given the ending sentence token in the previous sentence).

Note 2: For a history word *h*, what if there are not any bigrams that occur once? This means that when you use ML estimate for P(w|h), for all the words following *h*, then there is no probability mass taken out for the unseen bigrams and your α(h) will be 0.
For a large data set, we don't expect this to happen. And if it happens, we often change to another smoothing method.
In this assignment, please do the following when this happens (i.e., no singleton bigrams for a history word *h*):
for all the maximum likelihood estimation P(w|h), use a discount factor γ (e.g., γ =0.99), $P'(w|h)= \gamma * P_{ML}(w|h)$;

then for α (h), the denominator remains the same, and 1- γ will be your numerator. The backoff part is the same as before, $P_{katz}(w|h)=\alpha(h) * P_{ml}(w)$.

Note 3: Your LM model file only contains the seen bigram probabilities. The probabilities for unseen bigrams will be computed on the fly during testing.

2. Testing.

Implement a program that reads in a test file and a LM file (contains the smoothed probabilities and the backoff weights from the training program), and reports the perplexity of the test set with respect to the model. When you see a bigram with zero count (not appearing in your LM file), use Katz smoothing to get the probability as described above.
Remember the perplexity of the test set is:

$$ppl(w_1^N) = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|h_i)}$$

Note 1: the perplexity value is calculated for the entire test set (i.e., N in the formula above is the number of tokens in the test set).

Note 2: similar to training, don't use bigrams crossing sentence boundaries. When you calculate the probability of a sentence, just use $p(w_1|<s>)\ldots\ldots p(</s>|w_n)$.

To simplify your program, in this homework we will use a test set that doesn't have any out-of-vocabulary words.

How to turn in your programming assignment?

1) Package your source files and submit your source code and executable for the training and testing steps, with proper readme files.

We may test your program using different data sets (it will be in exactly the same format as the files provided to you).
You don't need to turn in your LM (it is too big), we will use your code to train the LM and then test it.

Your executable should look like this:

bigram-train -text training_file -lm lm_file
bigram-test -text test_file -lm lm_file (print to standard output or add another option to print the perplexity value to a file)

2) submit your report, with the experiment results on the test set, and any findings you have regarding the LMs.

In addition to the perplexity results required, you may consider looking at some other factors, for example, the effect of the training data size, perplexity difference between using a unigram and bigram, etc.

Please remember to put your name in your report.

Extra credits:

You can get extra credits (or just for fun) to implement the following programs.

- Use different smoothing methods.
- Use a trigram model.
- Deal with the out-of-vocabulary words during testing (you also need to think about what you need to do during training)