Assignment 1: Language Modeling

CSEP 517, University of Washington
Adapted from Dan Klein's CS288 at UC Berkeley
Due: Tuesday, October 15th

1 Setup

Download the assignment code and data from the CSEP517 share space, linked on the course website:

code1.zip : the Java source code.
data1.zip : the data sets

The authentication restrictions are due to licensing terms. You can login with you UW userid and password.

Check that you can compile the entirety of the code without errors (if you get warnings about unchecked casts, ignore them - that is a Java 1.5 issue). If you are at the source root (i.e. your current directory contains only the directory 'edu'), you can compile the provided code with

```
javac -d classes */*/*/*.java */*/*/*/*.java
```

You can then run a simple test file by typing

```
java -cp classes edu.berkeley.nlp.Test
```

You should get a confirmation message back.

For this assignment, the first Java file to inspect is:

src/edu/berkeley/nlp/assignments/LanguageModelTester.java

Try running it with:

```
java edu.berkeley.nlp.assignments.LanguageModelTester -path DATA -model baseline
```

where DATA is the directory containing the contents of the data zip.

If everythings working, you'll get some output about the performance of a baseline language model being tested. The code is reading in some newswire and building a basic unigram language model that we've provided. This is very bad language model. Your job is to improve it.

2 Assignment Description

In this assignment, you will construct several language models and test them with the provided code (main method in LanguageModelTester.java).

Training: First, LanguageModelTester.java loads about 1M words of WSJ text. These sentences have been "speechified" (for example translating "$" to "dollars") and tokenized. The WSJ data is split into training data (80%), validation (held-out) data (10%), and test data (10%). In addition to the WSJ text, the code loads a set of speech recognition problems (from the HUB data set). Each HUB problem consists of a set of candidate transcriptions of a given spoken sentence. For this assignment, the candidate list always includes the correct transcription and never includes words unseen in the WSJ training data. Each candidate transcription is accompanied by a pre-computed acoustic score, which represents the degree to which an acoustic model matched that transcription. These lists are stored in SpeechNBestList objects. Once all the WSJ data and HUB lists are loaded, a language model is built from the WSJ training sentences (the validation sentences are ignored entirely by the provided baseline language model, but may be used by your implementations for tuning). Then, several tests are run using the resulting language model.

Evaluation: Each language model is tested in two ways. First, the code calculates the perplexity of the WSJ test sentences. In the WSJ test data, there will be unknown words. Your language models should treat all entirely unseen words as if they were a single UNK token. This means that, for example, a good unigram model will actually assign a larger probability to each unknown word than to a known but rare word - this is because the aggregate probability of the UNK event is large, even though each specific unknown word itself may be rare. To emphasize, your model's WSJ perplexity score will not strictly speaking be the perplexity of the exact test sentences, but the UNKed test sentences (a lower number).

Second, the code will calculate the perplexity of the correct HUB transcriptions. This number will, in general, be worse than the WSJ perplexity, since these sentences are drawn from a different source. Language models predict less well on distributions which do not match their training data. The HUB sentences, however, will not contain any unseen words.

Third, the code will compute a word error rate (WER) on the HUB recognition task. The code takes the candidate transcriptions, scores each one with the language model, and combines those scores with the pre-computed acoustic scores. The best-scoring candidates are compared against the correct answers, and WER is computed. The testing code also provides information on the range of WER scores which are possible: note that the candidates are only so bad to begin with (the lists are pre-pruned n-best lists). You can inspect the errors the system is making on the speech re-ranking task, by running the code with the -verbose flag.

Finally, the code will generate sentences by randomly sampling your language models. The provided language models outputs arent even vaguely like well-formed English, though yours will hopefully be a little better. Note that improved fluency of generation does not mean improved modeling of unseen sentences.

2.1 Part 1 - Basic Model

As a first step, implement a simple n-gram language model that allows n to vary from two to four. Unigrams have been implemented in edu.berkeley.nlp.assignments.EmpiricalUnigramLanguageModel. You should use the provided code to evaluate your model, and the code you write should be able to estimate the probability of a given sentence and generate a new sentence (see edu.berkeley.nlp.langmodel.LanguageModel). Take a look at the type of errors your language model is making. The next two parts of the assignment are designed to target these errors.

2.2 Part 2 - Linear Interpolation

After building the simple model, your next task is to build a model that uses linear interpolation. Your model should again be able to set n from two to four. In this task, you will need to decide how to set the parameters (i.e. $\lambda_1, \lambda_2, \lambda_3$). See the Collins notes and lecture slides for some suggestions.

2.3 Part 3 - Discounting

The last model you will write uses discounting, and again should work for n from two to four. As in the previous task, you will need to decide how to set the parameter B. The notes and slides have suggestions for doing this as well.

2.4 Write-up

After writing each of the models described above, you should design and perform experiments to (1) compare the basic model with each of the smoothed models and (2) compare the smoothed models with each other.

Your write-up should be **no more than 4 pages long**, and should include:

1. A brief summary of how you set the parameters in the models you implemented and any

other significant choices you made.

2. A description of your experiments. This section should be brief, but with enough detail to allow us to reproduce your experiments. Be careful to design an experimental setup that measures generalization for the learned models, using the training, dev, and test sets appropriately as described in class.

3. Tables or graphs of the perplexities, accuracies, etc. to support all of your claims.

4. Error analysis – what are the systems doing wrong and how you might fix those problems. Some questions you might consider: When you do see improvements in WER, where are they coming from, specifically? Are there some models that perform better on perplexity but do not show an improvement in WER? You should do some data analysis on the speech errors that you cannot correct. Are there cases where the language model isn't selecting a candidate which seems clearly superior to a human reader? What would you have to do to your language model to fix these cases? Do certain models generate better text? Why? You do not need to answer all of these questions. Just do enough error analysis to convince us that you know which models work best, and why.

2.5 What to Turn in

You should turn in your write-up (as a pdf file) and your code (in a single compressed file including a README with instructions on how to replicate your reported experiments). Please submit your work online to the class DropBox, linked from the website.