cs294-5: Statistical Natural Language Processing
Assignment 1: Language Modeling

Due: Sept. 19th

Setup: On your server set of choice, make sure you can access the following directories under /home/ff/cs294-5 or /work/cs294-5 (ROOT from here on):

src/edu/berkeley/nlp/ : the code provided for this course
corpora/assignment1 : the data sets used in this assignment

Copy the source files to your local java/src directory. Some of the files and directories won't be relevant until later assignments, but feel free to poke around. Make sure you can compile the entirety of the course code without errors (if you get warnings about unchecked casts, ignore them – that's a Java 1.5 issue). The important Java file to start out inspecting is:

assignments/LanguageModelTester.java

Try running it with:

java edu.berkeley.nlp.assignments.LanguageModelTester –path ROOT/corpora/assignment1 –model baseline

If everything's 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 language model that I've provided. This is phenomenally bad language model, as you can see from the strings it generates – you'll improve on it.

Language Modeling: In this assignment, you will construct several language models and test them with the provided harness.

Take a look at the main method of LanguageModelTester.java, and its output. It loads several objects. First, it loads a training corpus of WSJ sentences, as well as a validation (held-out) set of sentences and a test set from the same source (split 80% / 10% / 10%). Second, it loads a set of speech recognition problems (from the HUB set). Each HUB problem is a set of candidate transcriptions of a spoken sentence, including the correct transcription. The candidates are each accompanied by an acoustic score which represents the degree to which the acoustic model matched the utterance with that transcription. These lists are stored in SpeechNBestList objects. Once these lists are loaded, a language model is built from the training sentences (the validation sentences are ignored entirely by the provided language model, but may be used by your implementations). Then, several tests are run using the constructed language model.

First, the tester calculates the perplexity of the test WSJ sentences. Then, the perplexity of the correct HUB answers is calculated. This number will in general be worse, since these sentences are drawn from a different source, newswire-trained language models will be worse at predicting them. Note that language models can treat all entirely unseen words as if they were a single UNKNOWN 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 probability of seeing some unknown word is large, even if each unknown word itself may be rare.

The third number produced is a word error rate (WER). The code takes the speech recognition problems' candidate answers, scores each candidate with the language model, and combines that score with the pre-computed acoustic score. 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: the correct answer is not always on the candidate list, and the candidates are only so bad to begin with (the lists are pre-pruned n-best lists). Note that if you want to see the errors the system is making on the speech re-ranking task, you can run the harness with the "-verbose" flag.

The final type of output is the result of generating sentences by randomly sampling the language models. Note that the provided language model's outputs aren't even vaguely like well-formed English, though yours will hopefully be a little better.

Your task is to implement several language models, though you have a good amount of choice of what specific ones to try out. Along the way you must build the following:

- A model employing either a Good-Turing estimator or a held-out estimator
- A higher-order model (at least a bigram model) which uses an interpolation method of your choice
- A method which makes some use of the held-out data set in some way

While you are building your language models, it may be that lower perplexity, especially on the HUB sentences, will translate into a better WER, but don't be surprised if it doesn't. The actual performance of your systems does not directly impact your grade on this assignment, though I will announce students who do particularly interesting or effective things.

What will impact your grade is the degree to which you can present what you did clearly and make sense of what's going on in your experiments using thoughtful error analysis. When you do see improvements in WER, where are they coming from? Try to localize the improvements if possible. Do the errors that are corrected by a given language model make sense? Are there changes to the models which substantially improve perplexity without improving WER? Do certain models generate better text? Why? Similarly, 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? For these kinds of questions, it's actually more important to sift through the data and find some good ideas than to implement those ideas. The bottom line is that your write-up should include concrete examples of errors or error-fixes, along with commentary.

Write-ups: For this assignment, you should turn in a write-up of the work you've done, as well as the code. Your code doesn't have to be beautiful or glowing with comments, but I should be able to scan it and figure out what you did without too much pain. The write-up should specify what models you implemented and what significant choices you made. It should include tables of the perplexities, accuracies, etc., of your systems. It should also include some error analysis – enough to convince me that you looked at the specific behavior of your systems and thought about what it's doing wrong and how you'd fix it. There is no set length for write-ups, but a ballpark length might be 3-6 pages, including your evaluation results, a graph or two, and some interesting examples. I'm more interested in knowing what observations you made about the models or data than having a reiteration of the formal statements of the various models.

Random Advice: In edu.berkeley.nlp.util there are some classes that might be of use – particularly the Counter and CounterMap classes. These make dealing with word to count and history to word to count maps much easier.