Statistical NLP 2016 - Assignment 1

due September 13 - at 5pm by email to petrov@cs.nyu.edu

This assignment is just for warming up. You will not need to write much code. Instead, I want you to familiarize yourself with the course logistics. Assignment 2 next week will be the first proper assignment.

1 Course Rules / Cheating Policy

I understand that most students would never consider cheating. There is, however, a fraction of students for whom this is not the case. To make sure we have a common understanding of what the course rules are, I ask you to print the last page of this assignment and acknowledge the rules by signing it. Please hand in your signed copy in class.

2 Git and Leaderboard

On the course webpage there is a leaderboard for tracking the prediction performance of the models that you will be building during the assignments. We will be using public GitHub repositories to manage the submissions. If you are not familiar with *git*, there are plenty of good tutorials online, for example: `http://www.vogella.com/articles/Git/article.html`.

If you don't already have a GitHub account yet, please create one here: `https://github.com/signup/free`. You are welcome to reuse an existing account for this course.

Now fork the master repository `https://github.com/slavpetrov/stat-nlp-nyu`. We will be mainly using the repository for submitting system output, so you will find a skeleton structure for `hw0` and `hw1`.

Please edit `hw0/output.txt` so it contains your name, email address, and a screen name for the course leaderboard and submit it via: `git add hw0/output.txt; git commit -m 'my first check-in'; git push;`

Finally, post your repository to Piazza, so that I can make sure it is included in the leaderboard. Your repository path should be of the form: `https://github.com/username/stat-nlp-nyu`. There is a manual step that I need to perform after you post your repository to Piazza, so please give me a couple of hours. After that everything will be automatic and the leaderboard will update every 5min.

3 Code Setup

Please send me an email so that I can point you to the Java source code for the course and the data sets needed for this assignment. This is necessary due to licensing restrictions.

Unzip the source files to your local working directory. Some of the classes and packages 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. If you cannot get the code to compile, email me or post to the newsgroup. If you are at the source root (i.e. your current directory contains only the directory 'nlp'), create a directory called 'classes'. You can then compile the provided code with

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

While you can certainly do everything from the command-line and using your favorite text editor, I would recommend using an IDE. Eclipse is easy to setup and makes writing code and debugging it a lot easier. Next, unzip the data into a directory of your choice. For this assignment, the first Java file to inspect is:

```
src/nlp/assignments/LanguageModelTester.java
```

Try running it with:

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

where DATA is the directory containing the contents of the data zip. If everythings working, you will 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 I have provided. This is phenomenally bad language model, as you can see from the strings it generates - you will improve on it.

4 Description

Now that the logistics are all settled, you can experiment with several language models and test them with the provided code framework. Take a look at the main method of LanguageModelTester.java, and its output.

Training: Several data objects are loaded by the harness. First, it loads about 1M words of WSJ text (from the Penn treebank, which we will use again later). These sentences have been ”speechified”, for example translating ”$” to ”dollars”, and tokenized for you. 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 harness 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 harness calculates the perplexity on two different held-out sets. This number can be useful for evaluation, but can be misleading. If you have a bug (and your probabilities do not sum to 1), you will get wonderful perplexities (but potentially terrible word error rates). If you don’t have any bugs, higher order n-gram models and more sophisticated smoothing techniques will result in lower perplexities.

The harness will first calculate 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). The harness will then 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.

More importantly, the harness will also 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 should inspect the errors the system is making on the speech re-ranking task, by running the harness with the -verbose flag.

Finally, the harness will generate sentences by randomly sampling from you 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.

Experiments: I have provided you with the implementation of several language models. You are welcome to implement your own additional language model (for example with Kneser-Ney smoothing), but you don't have to. Check out the models that I have provided you, evaluate their performance and write-up your findings. As you increase the complexity of your language model, 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.

To submit results to the leaderboard, edit `hw1/output.txt` and replace the GUESS and GOLD lines with what the harness prints when you set the -verbose flag. You can then push your changes to the repository and the leaderboard will automatically update shortly after. Make sure that your submission is recorded. If it doesn't appear on the leaderboard for several hours, please send me an email.

With an interpolated bigram model ($\lambda = 0.6$) you should be able get the word error rate down to 7.3% with a perplexity of around 480. An interpolated trigram model reduces the word error rate further to 6.9%. In both cases I simply reserved 1/totalCount for words not in the vocabulary.

Random Advice: In 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.

5 Write-ups

What will impact your grade is the degree to which you can present what you did clearly and make sense of what is going on in your experiments using thoughtful error analysis. When you do see improvements in WER, where are they coming from, specifically? Try to localize the improvements as much as possible. Some example questions you might consider: Do the errors that are corrected by a given change to the language model make any 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 is not 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 is 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.

For this assignment, you should turn in a write-up of the work you have done. The write-up should specify what models you implemented and what significant choices you made. It should include tables or graphs 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 is doing wrong and how you would fix it. There is no set length for write-ups, but a ballpark length might be 3-4 pages, including your evaluation results, some graphs, and some interesting examples. Since this is just the warm-up assignment, 2 pages should suffice. I am more interested in knowing what observations you made about the models or data than having a reiteration of the formal definitions of the various models.

While typesetting your write-up in Latex is not a requirement, it can be an useful exercise and will come in handy for later assignments (and the project) that might involve writing down some formulas.

You can email me your write-up in PDF format before the class on the due date or bring a print-out to class.

You should also submit your code to the git repository. Your grade will be based on your write-up, but I might also take a look at your code. It can also sometimes be useful to mention code choices or even snippets in write-ups – feel free to do so if appropriate, but this is not necessary.

6 Course Rules / Cheating Policy

I understand that most students would never consider cheating in any form. There is, however, a fraction of students for whom this is not the case. To make sure we have a common understanding of what the course rules are, I ask you to print this page and acknowledge the rules by signing it. Please hand in your signed copy in class.

The rules below are adapted from Smith & Dyer at CMU (see their class Natural Language Processing (11-{4,6}11)).

- You may verbally collaborate on homework assignments. On each problem and program that you hand in, you must include the names of the people with whom you have had discussions concerning your solution. Indicate whether you gave help, received help, or worked something out together. The names should include anyone you talked with, whether or not they're taking the class, and whether or not they attend or work at NYU.
- You may get help from anyone concerning programming issues which are clearly more general than the specific assignment (e.g., "what does a particular error message mean?").
- You may not share written work or programs (on paper, electronic, or any other form) with anyone else.
- If you find an assignment's answer, partial answer, or helpful material in published literature or on the Web, you must cite it appropriately. Don't claim to have come up with an idea that wasn't originally yours; instead, explain it in your own words and make it clear where it came from.
- On the course project, you are encouraged to use existing NLP tools. You must acknowledge these appropriately in all documentation, including your final report. If you aren?t sure whether a tool or data resource is appropriate for use on the project, because it appears to solve a major portion of the assignment or because the license for its use is not clear to you, or if you aren't sure how to acknowledge a tool appropriately, you must speak with the course staff.

Clear examples of cheating include (but are not limited to):

- Showing a draft of a written solution to another student.
- Showing your code to another student.
- Getting help from someone or some resource that you do not acknowledge on your solution.
- Copying someone else's solution to an assignment.
- Receiving class related information from a student who has already taken the exam.
- Attempting to hack any part of the course infrastructure.
- Lying to the course staff.

I hereby acknowledge that I have read and understood the course rules.

Date:

Name:

Signature: