Statistical NLP 2017 - Assignment 1

due September 17 - at 11:59pm EST by email to aparikh@cs.nyu.edu and anhad@nyu.edu

This assignment is shorter than the rest. Assignment 2 will be the first proper assignment.

Deliverables:

- Writeup emailed to instructors. The recommended writeup length for this assignment is 2 pages. To give you some freedom to not worry about space saving, you may submit a report up to 4 pages (including any figures / charts). Reports above 4 pages will be penalized.
- Answers to problem set questions. If you use good handwriting, you are welcome to handwrite them and email us a scanned copy.
- Signed course rules / cheating policy handed in class (On September 11 or 18)

Submission Format: Please title your submission email **Assignment 1 submission**. You should submit a zip file (firstname_lastname_hw1.zip) containing the report in pdf format (firstname_lastname_hw1.pdf) and the zipped code directory. Write your netID and your collaborators' netID (if any) in the report.

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 Code Setup

Please send us both (Anhad and I) an email so that we 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 everything is 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.

3 Description

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%) (the code currently ignores the validation set, but in reality you would use it to tune hyperparameters etc.) 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. 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 combined with 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. The ones you need to experiment with for this assignment are:

- **unigram** - EmpiricalUnigramLanguageModel.java
- **bigram** - BigramEmpiricalLanguageModel.java
- **trigram** - TrigramEmpiricalLanguageModel.java

In particular, for the bigram and trigram models, note the interpolation weight variables in the code: *lambda*, *lambda1*, *lambda2*. These variables control the interpolation between the higher order and lower order ngrams.

I would like you to explore the following aspects of the above language models and write a ≈ 2 page writeup detailing your results and analysis.

- **12 points**: Effect of interpolation weights *lambda*, *lambda1*, *lambda2*.
- **12 points**: Relationship between WSJ and HUB performance e.g. are there models where the gap is larger/smaller than others?
- **12 points**: Relationship between HUB perplexity and HUB error rate e.g. do you see cases where the error rate goes down but the perplexity increases. Why do you think that is?

Please include graphs or charts detailing your experiments (e.g. how performance varies as λ is changed). You should also extract out general trends / possible explanations instead of just reporting numbers.

4 Mini-Problem Set

Language Modeling (12 points): Let n be the order of the language model i.e.

$$P(w_i|w_1, ..., w_{i-1}) \approx P(w_i|w_{i-n+1}, ..., w_{i-1}) = P(w_i|w_{i-n+1}^{i-1})$$

For this question, the term *inference time* refers to one call to the language model i.e. computing $P(w_i|w_{i-n+1}^{i-1})$ under the language model for one choice of w_{i-n+1}^{i-1}, w_i.

- Characterize the inference time complexity of Kneser Ney language models in big-O notation. You should decide what variables are important to model when computing these two aspects.
- Characterize the memory complexity of Kneser Ney language models in big-O notation. You should decide what variables are important to model when computing these two aspects.

5 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: