DD1418/DD2418 Language Engineering
2018-11-06

Assignment 2

Readings: Read chapter 3 in Jurafsky-Martin.

Code: The skeleton code can be downloaded from Canvas or from

`http://www.csc.kth.se/~jboye/teaching/language_engineering/a02/LanguageModels.zip`

Unzip the code in your home directory. Go to the folder `LanguageModels` and type:

```
pip install -r requirements.txt
```

Now everything needed for the assignment should be installed.

Problems:

1. We want a program that computes all bigram probabilites from a given (training) corpus, and stores it in a file. For instance, from the file `data/small.txt`:

   ```
   I live in Boston.
   I like ants.
   Ants like honey.
   Therefore I like honey too!
   ```

   we want to produce the contents of the file `small_model_correct.txt`. Note that:

   - The first line contains two numbers, separated by space: The vocabulary size V (= the number of unique tokens, including punctuation), and the size of the corpus N (= the total number of tokens).
   - Then follows V lines, each containing three items: an identifier $(0, 1, \ldots)$, a token, and the number of times that token appears in the corpus.
   - Then follows a number of lines, one for each non-zero bigram probability. Each line contains three numbers: The identifiers of the first and second token of the bigram, respectively, followed by the logarithm of the bigram probability, printed with 15 decimals. The natural logarithm is used (as computed by the `math.log` library method).
   - The final line is "-1" to mark end-of-file.

   The `BigramTrainer.py` program contains a skeleton program for reading a corpus, computing unigram counts and bigram probabilities, and printing the model. **Your task is to extend the code so that the program works correctly** (look for the comments `YOUR CODE HERE` in the program). Use the scripts `run_trainer_small.sh` and `run_trainer_kafka.sh` to run the program on test examples.

You can use the `-d` option to save the model to file, e.g. :

```
python BigramTrainer.py -f data/kafka.txt -d kafka_model.txt
```

If you are using Windows, printing the model to the terminal will likely lead to character encoding errors.

By adding the `--check` flag, you can verify that your results are correct.

```
python BigramTrainer.py -f data/kafka.txt --check
```

2. The `BigramTester.py` program contains a skeleton program for reading a model on the format described in the previous problem, reading a test corpus, and computing the entropy of the test corpus given the model (the cross-entropy of the training set and the test set).

   (a) Extend the code so that the program works correctly (look for the comments `YOUR CODE HERE` in the program). The entropy of the test set is computed as the average log-probability:

   $$-\frac{1}{N}\sum_{i=1}^{N}\log P(w_{i-1}w_i)$$

   where N is the number of tokens in the test corpus. To be able to handle missing words and missing bigrams, use linear interpolation:

   $$P(w_{i-1}w_i) = \lambda_1 P(w_i|w_{i-1}) + \lambda_2 P(w_i) + \lambda_3$$

   The values for the constants λ_1, λ_2 and λ_3 are given in the code for the `BigramTester` program. The script `run_tester_small_kafka.sh` tests the model built from `small.txt` using `kafka.txt` as a test corpus, and the script `run_tester_kafka_small.sh` tests the model build from `kafka.txt` on the test corpus `small.txt`. Compare with my numbers by using the `--check` flag. (Your numbers might deviate slightly from my numbers; for instance, if you are using a different logarithm. I used the natural logarithm.)

   (b) Build a model from the file `data/guardian_training.txt` and another model from `data/austen_training.txt`. Compute the entropy of the test file `guardian_test.txt` and the test file `austen_test.txt`, using both models. Report your numbers and your conclusions from these experiments!

3. (Only compulsory for DD2418) It is also possible to generate words according to the probabilities of a language model. If, for example, the last generated word was `red`, and according to the model `red` can be followed by one of the words `ball` with $p = 0.5$, `toy` with $p = 0.3$, or `car` with $p = 0.2$, then the next generated word should be either `ball`, `toy`, `car` with probabilities 0.5, 0.3 and 0.2, respectively.

   The `Generator.py` program contains a skeleton program for generating words from a language model in the way just described. Extend the code so that the program works correctly (look for the comments `YOUR CODE HERE` in the program). Then generate some words from the various models you have constructed in the preceding problems. (In the rare event where all bigram probabilies from the last generated word are zero, then pick any word at random using a uniform distribution).