FNLP: Assignment 1

Corpora and Language Identification

Deadline: Thursday 14 February 2019 1600

1 Introduction

This assignment will make use of the Natural Language Tool Kit (NLTK) for Python. NLTK is a platform for writing programs to process human language data, that provides both corpora and modules. For more information on NLTK, please visit: `http://www.nltk.org/`.

Please note that this assignment is for **formative feedback** only (see below for further information).

Getting Started

Before continuing with this assignment, please download a copy of the assignment template[1]. This package contains a code template and associated libraries that you *must* use as a starting point when attempting the questions for this assignment.

Submitting Your Assignment

Submit your assignment code using the submit system:

```
$ submit fnlp 1 assignment1.py
```

Before submitting your assignment:

- Ensure that your code works on DICE. Your assignment must work when the following command is run under the **fnlp** environment:

  ```
  (fnlp)... $ python assignment1.py
  ```

  See the lab introduction for instructions on how to enable the **fnlp** environment.

- Test your code thoroughly. If your code crashes when run, up to 25 marks may be deducted for code that fails to run (on DICE, under the **fnlp** environment, using python3), depending on the severity of the coding error(s).

[1] `http://www.inf.ed.ac.uk/teaching/courses/fnlp/coursework/assignment1.tar.gz`

- Ensure that you include comments in your code where appropriate.
- **Important:** Whenever you use corpus data in this assignment, you *must* lowercase the data, so that e.g. the original tokens “Freedom” and “freedom” are made equal. Do this *throughout* the assignment, whether it’s explicitly stated or not.

Late Coursework Policy

You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

If you submit anything before the deadline, you may not resubmit afterward. (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, then you won’t get the formative feedback from this assignment unless you have received an approved extension.

Warning: Unfortunately the `submit` command will technically allow you to submit late even if you submitted before the deadline (i.e. it does not enforce the above policy). Don’t do this! We will mark the version that we retrieve just after the deadline, and (even worse) you may still be penalized for submitting late because the timestamp will update.

For additional information about late penalties and extension requests, see the School web page below. Do **not** email any course staff directly about extension requests; you must follow the instructions on the web page.

`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projec`
`late-coursework-extension-requests`

Good Scholarly Practice

Please remember the University requirement as regards all assessed work. Details about this can be found at:

`http://www.ed.ac.uk/academic-services/students/conduct/academic-misconduct`

and at:

`http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.`
`html`

Section A: Comparing corpora (10 marks)

Question 1 (5 Marks)

a. Find the average word length (averaging over distinct tokens, a.k.a word *types*) for the whole Presidential inaugural speeches corpus. This is the same inaugural corpus that you used in Lab 1.

b. Find the average word type length for the Twitter data posted on 28/01/2010 (`20100128.txt`) and 29/01/2010 (`20100129.txt`). Use the tweets from both days and take a single average. Return a float value.

   You will need to use the `xtwc` corpus. Access to this corpus is provided by the `twitter` module.

In both cases you should use the default NLTK tokeniser, by using the `.words()` method, and print the (floating point) average value to two significant digits.

Also, don't forget to lowercase your data (we won't remind you again).

Question 2 (5 Marks)

Is the average word type length greater for the Twitter data or the Presidential inauguration speeches? Why do you think this is? Give examples where appropriate.

There is a 500 character limit on answers to this question.

Your answers do not need to be complete sentences.

Section B: Data in the real world (30 marks)

Question 3 (10 Marks)

Compute frequency distributions and use them to identify the most frequent 50 types in the inaugural and Twitter (28/01/2010 and 29/01/2010) corpora. For each corpus, compute the top 50 types and their frequencies, then plot and return them.

Question 4 (5 Marks)

As is typical for real world data, the Twitter corpus is rather noisy. It contains a lot of non-alphanumeric tokens, which should be removed so that we can obtain more meaningful results. The inaugural corpus, whilst already reasonably clean, contains a lot of function words and punctuation marks, which aren't very informative. We would like to remove all function word tokens as well as tokens that contain only punctuation marks.

Write a function that takes in a list of all tokens in a corpus and:

a. Removes all stopword tokens, using the English stopwords list provided in `nltk.corpus.stopwords`;

b. Removes all non-alphanumeric tokens, using Python's `.isalnum()` string method.

Use this function to produce a cleaned token list for each corpus: Twitter and inaugural. Print out the lengths of each corpus, before and after cleaning, and the first 100 tokens in the cleaned version.

Question 5 (5 Marks)

Using the cleaned token lists from Question 4, re-compute the frequency distributions as you did for Question 3. For each corpus, compute the top 50 types and their frequencies, plot and return as before.

Question 6 (10 Marks)

The Twitter data is still very noisy despite having removed English stopwords and non-alphanumeric strings. What else could be done to clean up the data so that it would be easier to work with? Please describe in detail the problems that you have identified with the data raw data, the extent to which the cleaning implemented in Question 5 has helped *and* further techniques that you could use for the cleaning up process. Give examples where appropriate.

N.B. You are not required to implement your suggestions.

There is a 1000 character limit on answers to this question.

Your answers do not need to be complete sentences.

Section C: Language identification (60 marks)

Question 7 (10 Marks)

Train a bigram letter language model using the Brown corpus from *nltk.corpus* and the `LgramModel` class in the `twitter` module. Write a function that performs the following steps:

a. Creates a list of all alpha-only tokens in the Brown corpus, using Python's `.isalpha()` string method;

b. Trains a padded bigram letter language model using the resulting cleaned data and returns it.

Remember, `help(...)` is your friend, and you can always "view source".

You may want to look at the `LgramModel` class in the `twitter` module, as well as its superclass `NgramModel`. For this question, the arguments to `LgramModel` should give the order of the desired model (n), the training data, which should be a list of words (analogous to training a word n-gram model on a list of sentences), and the necessary flags to ensure padding both before and after the samples. The default estimator will provide smoothed estimates, don't change it.

Finally, note that the Brown corpus is rather large so training the language model may take a minute.

Question 8 (15 Marks)

Clean up the data in the first file of the tweet corpus (`20100128.txt`) to remove all non-alpha tokens and any tweets with fewer than 5 tokens remaining (i.e. after token removal). Given the bigram letter language model that you trained on the Brown corpus in Question 7, compute the **average word cross-entropy** of the words in each tweet in the resulting test set. That is, for each tweet you should compute the per-letter cross-entropy of each word, then take the average of that value over the words in the tweet. Return a list of tuples of the form: `[(entropy_1,tweet_1), ..., (entropy_n, tweet_n)]` where `n` is the total number of tweets in the test set, sorted in ascending order of average word cross-entropy value.

See `help(NgramModel.entropy)`. Use the correct arguments to ensure padding and to get the per-letter entropy (as opposed to the total).

If you want to see the code for the `LgramModel` and `NgramModel` classes, they are in `twitter/twitter.py` and
`nltk_model/ngram.py` respectively.

Question 9 (10 Marks)

Look at some of the tweets with the best (top 10% in the list) and worst (bottom 10%) cross-entropy values. What do you notice? State what you observe and explain the roles of the bigram letter language model and average word cross-entropy values in obtaining the result that you observed.

There is a 500 character limit on answers to this question.

Your answers do not need to be complete sentences.

Note: what you will see are anonymised tweets – the words in the tweets have been re-ordered and usernames have been replaced by strings of numbers. We were required to do this as a condition of using the Twitter data. You do not need to reconstruct the correct token order of the tweets.

Question 10 (15 Marks)

Now we will do some more cleaning to remove tweets likely to have a lot of non-ascii characters and then try to separate out those ‘ascii’ tweets that probably aren’t written in English:

Your function should accept a list of tuples of the form: `[(entropy_1, tweet_1), ..., (entropy_n, tweet_n)]`.

a. Set a threshold value to remove the bottom 10% of tweets (these are probably non-ascii tweets) as these are almost definitely not English tweets. The input should be the sorted tweet cross-entropy list computed in Question 8. Call these the ‘ascii’ tweets (and their cross-entropies).

b. Use this thresholded list to compute the mean and standard deviation of the cross-entropy values. You may use the `numpy` module for these computations.

c. Now subset the thresholded list to get only those with a cross-entropy greater than `mean + (0.674 * standard_deviation)` (for a normal distribution, $\mu + 0.674\sigma$ corresponds to the 75th percentile). We’ll call these the ‘probably not English’ tweets. The list should be still be sorted in ascending order of cross-entropy value.

The function should return 4 values, as follows:

a. The mean from part b above;

b. the standard deviation from part b;

c. the ‘ascii’ tweets and entropies from part a;

d. the ‘probably not English’ tweets and entropies from part c.

Question 11 (10 Marks)

Now that we have a list of ‘probably not English’ tweets, let’s find out whether there are any Spanish tweets in that list. Train a bigram letter language model using files `esp.test` and `esp.train` from the Spanish portion of the CoNLL 2007 corpus: `nltk.corpus.conll2007`. Preprocess the data similarly to the way you did the Brown data, that is, down-case each token, and discard any that are non-alpha.

The CoNLL 2007 corpus consists of dependency treebanks in a number of languages and was originally compiled for a shared task on dependency parsing, hence the provision of a split corpus consisting of testing and training data. The Spanish portion of the corpus contains a collection of texts of multiple different genres, in Spanish, Basque and Catalan.) Re-compute the entropy of each tweet in the ‘probably not English’ tweets list (from Question 10) using the Spanish model. Return a sorted list of pairs of entropy and tweet as before.

The whole CoNLL corpus is very large and contains data in other languages. Be sure to restrict your selection to `esp.test` and `esp.train` for training the Spanish bigram letter language model.

Print the tweets at the top and bottom of this list (i.e. 10 tweets at each end of the list) and confirm that the top looks to have more Spanish words than the bottom.