FNLP: Assignment 1

Corpora and Language Identification

Deadline: 4pm, 15th February 2018

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: `www.nltk.org`.

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

1.1 Getting Started

Before continuing with this assignment, please **download a copy of the assignment template** (assignment1.py) from the FNLP course website. **This template contains code that you must use as a starting point when attempting the questions for this assignment.**

1.2 Submitting Your Assignment

Your submission will be a python file. Please name it with your matriculation number: e.g., s0123456.py. Then submit your assignment code using the submit system:

$ submit fnlp cw1 s0123456.py

Before submitting your assignment:

- Ensure that your code works on DICE. Your assignment must work when the following command is run: $ python s0123456.py
- Test your code thoroughly. If your code crashes when run you may be awarded a mark of 0. If your code does not run to completion then it will

make it very difficult for us to test your answers, we cannot be expected to fix your code while marking.

- Ensure that you include comments in your code where appropriate. This makes it easier for the markers to understand what you have done and makes it more likely that partial marks can be awarded.
- Any character limits will be strictly enforced. Answers will be passed through an automatic filter that only keeps the first x characters, where x is the character limit. Therefore, anything after the character cut-off will not be read.

Note that the feedback you get for this course does not contribute to your overall grade! You will get **formative** feedback from this assignment consisting of: (a) a mark out of 100; and (b) qualitative feedback that helps you understand why you got the mark you did.

1.3 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-projects/late-coursework-extension-requests`

1.4 Good Scholarly Practice

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

`http://www.ed.ac.uk/schools-departments/academic-services/students/undergraduate/discipline/academic-misconduct`

and at:

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

Section A: DATA IN THE REAL WORLD (45 Marks)

Question 1 (15 Marks)

Compute frequency distributions and use them to identify the most frequent 50 tokens in the inaugural and Twitter (28/01/2010 and 29/01/2010) corpora. For each corpus, return a **list** of the top 50 tokens and their frequencies. Then plot the top 50 tokens against their frequencies.

Question 2 (20 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 "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.
Hint: use the ***English stopwords*** *list provided in* ***nltk.corpus.stopwords****.*

b) Removes all non-alphanumeric tokens.
Hint: use Python's ***.isalnum()*** *string method.*

Use this function to produce a "cleaned" token list for each corpus: Twitter and inaugural.

Using the "cleaned" token lists, re-compute the frequency distributions as you did for Question 1. For each corpus, return a **list** of the top 50 tokens and their frequencies. Then plot the top 50 tokens against their frequencies.

Question 3 (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 **and** the techniques that you could use for the cleaning up process. Give examples where appropriate.

There is a 280 character limit on this question.

Your answers do not need to be complete sentences.

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

Section B: LANGUAGE IDENTIFICATION (55 Marks)

Question 4 (15 Marks)

Train a bigram letter language model using the **Brown** corpus in **nltk.corpus** and **LgramModel** in the **twitter.py** module (in /group/ltg/projects/fnlp). Write a function that performs the following steps:

a) Creates a list of all alpha-only tokens in the Brown corpus.
Hint: use Python's ***.isalpha()*** *string method.*

b) Trains a bigram letter language model using the "cleaned" Brown data from step "a".
Hint: Look at the LgramModel code in the twitter module. For this question, LgramModel should be called with only two arguments: order of the model (n) and the training data.)

c) Returns the trained LgramModel.

General hint: the Brown corpus is rather large so training the language model will take some time. LgramModel collects a letter n-gram model from a list of strings. It is a minor tweak of Python's NgramModel.

Question 5 (25 Marks)

Clean up the tweet corpus 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 in Question 5, compute the *average word entropy* of each tweet in the *cleaned* test set **xtwc.sents("20100128.txt") from the twitter module**. 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. The list should be sorted in ascending order of average word entropy value.
*Hint: remember you have a bigram **letter** language model and the tweets are lists of words. You will need to compute the entropy at the word level and then normalise for "sentence" length.*

Question 6 (15 Marks)

Now we will do some more cleaning to remove non-ascii tweets and those ascii tweets that probably aren't written in English:

a) Set a threshold value to remove the bottom 10% of tweets (the non-ascii tweets) as these are definitely not English tweets. The input should be the tweet entropy list computed in Question 8. The input list should be sorted in ascending order of entropy value. Return the list of "ascii" tweets (and their entropies).

b) Use this "thresholded" list (from part a) and compute the mean and standard deviation for the entropy values. Your function should accept a list of tuples of the form: [(entropy_1, tweet_1),...,(entropy_n, tweet_n)]. Return the mean and standard deviation values as **float** values.
*Hint: Consider using the **numpy.py** module for these computations.*

c) Now collect those tweets from the list of "ascii" tweets with an entropy greater than *(mean + (0.674 * standard_deviation))* into a new list of tuples of the form: [(entropy_1, tweet_1),...,(entropy_n, tweet_n)]. These will be the "probably not English" tweets. The list should be sorted in ascending order of entropy value. Return the list of "probably not English" tweets (and their entropies).