1 TweetNLP and Language Complexity

The goal of this assignment is to introduce some of the challenges of working with user-generated content. You will first review a paper and then apply the code developed by its authors to Twitter data. You will use the outputs to find tweets that carry complex language.

1.1 Paper review (40%)

Read the following research paper in-depth and write a **2-page** summary (or review) of the paper. The review should provide a concise summary of the work including the problem, main contributions, and (perceived) strengths and limitations of the paper:

- Gimpel, K., et al. *Part-of-speech tagging for twitter: annotation, features, and experiments*. ACL'11.
  paper: `https://aclanthology.org/P11-2008.pdf`
  code: `http://www.cs.cmu.edu/~ark/TweetNLP/`

The review should include the following sections:

- *Problems and contributions: (5%)* Summarize the problem of interest and (theoretical, methodological, algorithmic or empirical) contributions of the paper.
- *Method: (10%)* Explain the key aspects of the proposed methodology.
- *Strengths: (5%)* Describe the strengths of the paper in terms of theoretical or empirical soundness as well as significance of the contributions.
- **[G]** *Limitations: (20%)* Explain the limitations of the paper; provide specific and detailed points, and avoid vague or subjective opinions. See Hints below.

1.2 Tokenization and POS Tagging (20%)

Use the TweetNLP toolkit (`http://www.cs.cmu.edu/~ark/TweetNLP/`) to tokenize and tag around 4k alcohol-related tweets (see the dataset `alcohol_tweets_4k.tar.xz` on Blackboard). Submit a text file named `processed_data.txt` in the following format:

```
original_tweet  TAB  tokenized_tweet  TAB  postagged_tweet
original_tweet  TAB  tokenized_tweet  TAB  postagged_tweet
...
```

where `original_tweet` is the text of a tweet, `TAB` is a the tab character, and `tokenized_tweet` and `postagged_tweet` are the outputs of the tokenizer and the part-of-speech tagger respectively. Your output file should have one line per tweet.

1.3 Tweet Complexity (40%)

User-generated content can be complex in terms of vocabulary and grammatical structures. Lexical complexity metrics inform natural language processing (NLP) systems in different applications such as those that quantify language proficiency or developmental level of second language (L2) learners. In this question, you will use the output of the TweetNLP toolkit to find *lexically* complex tweets in your data. A careful lexical complexity analysis will give you ideas for the Limitations question in the Paper Review section.

a. Density: Generate a *ranked* list of tweets based on the ratio of the number of *open-class* words (n_{lex}) to the total number of words (n) in each tweet. Note that open-class words are *nouns*, *adjectives*, *verbs* and *adverbs*. Submit a text file named `density.txt` in the following format:

```
original_tweet  TAB  n_lex/n
original_tweet  TAB  n_lex/n
...
```

where n_{lex}/n is a real value indicating the lexical density ratio of the corresponding tweet. Tweets should be ranked in descending order of their density ratios, and the output file should have one line per tweet.

b. Sophistication: Generate a *ranked* list of tweets based on the ratio of *sophisticated* words in each tweet. Sophisticated words are advanced words that are *not* frequently used in a language. We uploaded the list of top 5k most frequent words of English Twitter to Blackboard (see `top_5k_twitter_2015.tar.xz`), and you can assume sophisticated words are those that are *not* in this list. Compute the ratio of the number of sophisticated open-class words for each tweet, i.e. (n_{slex}/n). Submit a text file named `sophistication.txt` in the following format, ranked in descending order of the above ratio.

```
original_tweet  TAB  n_slex/n
original_tweet  TAB  n_slex/n
...
```

c. Diversity: Generate a *ranked* list of tweets based on the diversity of vocabulary in each tweet. As the diversity measure use *type-token* ratio (TTR) which is the ratio of the number of *unique* words (often referred to as word *types*) (s) to the number of words (n) in the tweet. Submit a text file named `diversity.txt` in the following format, where tweets are ranked in descending order of their TTRs:

```
original_tweet  TAB  s/n
original_tweet  TAB  s/n
...
```

d. Correlation Analysis: Use `SciPy` library to compute the *Pearson* correlation coefficient[1] between each pair of the above quantities. The Pearson correlation coefficient measures the linear relationship between two given arrays. For example, to compute the correlation between the density and sophistication rankings of tweets, create two arrays $\mathbf{a}=[a_0, a_1,\ldots, a_n]$ and $\mathbf{b}=[b_0, b_1,\ldots, b_n]$, where a_i and b_i indicate the density and sophistication of the ith tweet in your dataset respectively. The correlation can then be computed using the command `stats.pearsonr(`**`a`**`,`**`b`**`)`, which returns the tuple (r, ρ), where r is the Pearson correlation coefficient and ρ is the p-value. Submit a text file named `correlations.txt` that contains (a): a result Table in the following format, where each cell reports its corresponding (r, ρ) tuple, and (b): explanation of the resulting correlations. Note that the possible values for r are in [-1,+1] range, where 0 implies no correlation, and -1 or +1 imply exact linear relationship. See this link[2] for interpretation of (r, ρ) values.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html`

[2] `https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/how-to/correlation/interpret-the-results/`

	density	sophistication	diversity
density			
sophistication			
diversity			

Provide explanation after the table: ...

1.4 Lexical Diversity (Optional, Additional Credit of 50%)

The type-token ratio (TTR) in 4.c has been criticized for its sensitivity to the length of input text, as the ratio tends to decrease as the length of the input text increases. Several approaches have been proposed to address this issue. In particular, the moving average type token ratio (MATTR; Covington, 2007; Covington & McFall, 2010[3]) measures diversity by calculating TTRs for successive non-overlapping segments of each input text. The algorithm selects a window length of k tokens (words), and the TTR for tokens 1 to k is estimated. Then, the TTR is estimated for tokens 2 to $(k+1)$, then 3 to $(k+2)$, and so on for the entire sample. The final score is the average of the estimated TTRs. Another approach is the measure of textual lexical diversity (MTLD; McCarthy, 2005[4]), which employs a sequential analysis of an input text to estimate its diversity. Specifically, it calculates the TTR for increasingly longer parts of the sample. Every time the TTR drops below a predetermined value, a count (called the factor count) increases by one (1), and the TTR evaluations are reset. The algorithm resumes from where it had stopped, and the same process is repeated until the last token of the input text has been added and the TTR has been estimated. Then, the total number of words in the text is divided by the total factor count. Subsequently, the input text is reversed and another score of MTLD is estimated. The forward and the reversed MTLD scores are averaged to provide the final MTLD estimate. See further details in the paper cited above.
Implement one of the above approaches and submit a text file named `mattr.txt` or `mtld.txt` (depending on the algorithm you implement) in the following format:

```
original_tweet  TAB  mattr_score|mtld_score
original_tweet  TAB  mattr_score|mtld_score
...
```

Hints on "Limitations" in Question 1.1: Artificial Intelligence (AI) systems such as many NLP tools can make errors. Some interesting tokenization errors are cases where the tokenizer splits words that should be regarded as a single token (word), e.g. "Los Angeles" or "self sustaining" in `..a company in Los Angeles is building a self-sustaining city on Mars..`; hyphenated words that should be separated: `..the hold-him-back-and-drag-him-away maneuver..`; words with missing or spurious spaces: `..wer an after him qui ckly..` which should yield `{.. we, ran, after, him, quickly,..}`; or unusual tokens that should be recognized as single tokens such as `M*A*S*H` (a TV series). Similarly part-of-speech taggers may make errors, e.g., by mislabeling an adjective as a noun, adverb or even verb. A good way to discover limitations

[3]Covington, M.A. and McFall, J.D., 2010. Cutting the Gordian knot: The moving-average type–token ratio (MATTR). Journal of quantitative linguistics, 17(2), pp.94-100. Link: `https://www.tandfonline.com/doi/pdf/10.1080/09296171003643098?needAccess=true`

[4]McCarthy, P.M., 2005. An assessment of the range and usefulness of lexical diversity measures and the potential of the measure of textual, lexical diversity (MTLD). Link: `https://www.proquest.com/docview/305349212?pq-origsite=gscholar&fromopenview=true`

of TweetNLP is to find sample tweets that are poorly or wrongly tokenized or tagged. For this purpose, examine the *most* and *least* complex tweets that you found in question 1.3.

2 Important Instructions

You must submit a single zip file named `[STUDENTID].zip` that contains **all** the following files (except the optional file if you are not submitting answer to question 1.4) in its root directory:

1. `review.pdf`: a **2-page** `PDF` file that contains your review.
2. `processed_data.txt`: in the format described above.
3. `density.txt`: in the format described above.
4. `sophistication.txt`: in the format described above.
5. `diversity.txt`: in the format described above.
6. `correlations.txt`: in the format described above.
7. [optional] `mattr.txt` or `mtld.txt`: in the format described above.
8. `code.py`: a script to run your (ideally python) code to generate items 2–7 above.
9. `readme.txt` a readme file containing instructions for running `code.py`.

Good luck with the assignment!