Project 1 for CS421 – University of Illinois at Chicago

Mohit Haresh Adwani: madwan2@uic.edu
Shobhit Lamba: slamba4@uic.edu

----------Setup----------

Please install required libraries using the following commands:
pip install nltk
pip install stanfordcorenlp

The program requires Stanford coreNLP server running on port 9000. If your version of Stanford coreNLP doesn't work properly. Please use version 3.8.0 from here.
Steps to run the coreNLP server:

1. On command prompt, switch to the Stanford corenlp folder (Ex: stanford-corenlp-full-2017-06-09)
2. Run command with the quotes –
   'java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000'
3. Make sure it is running on port 9000

In the *execution* folder, give the following command on your command prompt:
python3 main.py

Since Stanford coreNLP is used, and the processing is done on the training data, the program takes significant amount of time to run. Because of this, we have included console print statement after it completes it's processing on each essay.

----------Technique----------

a) Length of Essay:

- First, we get the sentences by applying nltk sent_tokenize on the essay. Then we split all the sentences which has newline character (\n) into multiple sentences.
- We saw a pattern in the essays in which there wasn't a space after period (which denotes the end of the sentence) and hence the sent_tokenize didn't split the sentence into two sentences. Hence, we split the sentences into two if the character after period is alpha and before the period isn't period(.) since some essays had two or more consecutive periods to denote continuation. Also the sentence after the period should at least have 3 characters.
- While looking at multiple finite verbs in the sentence(hint given in project_part1.pdf), if the sentence didn't have coordinate or subordinate clause then we saw a pattern in the parse tree, denoting the finite verb phrase as '(SBAR (S' – where SBAR denotes 'clause introduced by a (possibly empty) subordinating conjunction'. If the sentence has a subordinating conjunction, then it is denoted by '(SBAR (IN that) (S'
- We noticed that splitting the sentence based on capitalization doesn't work well and there are very few sentences in the whole training data which can be split using capitalization, hence this method wasn't implemented.

b) Spelling Mistakes

We took a multilevel approach for counting number of spelling mistakes:

- First, we simply tried to find the spelling mistakes using wordnet. But wordnet falsely reported simple words like “how”, “you” etc as wrong spellings. So we used it as our first pass of spellcheck and its output was used as an input for the next step.
- After that, I employed Peter Norvig’s spell correction code to further filter out the words. Now to a great accuracy, we can say that the words left in the list are probably the only words that are incorrect in the essay.

c) Syntax/Grammar

Grammar is the hardest task we had to tackle in part 1 of the project. Here, we had to handle the following:

- ➔ Subject-Verb Agreement
- ➔ Missing Verbs
- ➔ Incorrect Verbs
- ➔ Verb Tense

- For this, we had to employ POS tagging. After every word of the essay was tagged, the task was subdivided as c.i and c.ii
- For c.i (Subject-Verb Agreement), we defined certain rules that a subject and verb combination shouldn’t follow, which were basically the singular and plural agreements. If the POS-tag sequences were found in the list of rules, error count is incremented.
- For c.ii (Rest of the conditions), we again defined rules, but this time the list included rules that are correct. So, if the sequence does not match any of the defined rules, error count is incremented.
    - o A list of four grams, tri grams, bi grams and uni grams POS tags were created based on the English verb formation rules.
    - o The program extracts the longest sequence of verb forming tags and checks whether it is present it out lists. If not, it is counted as an error.
- For c.iii (Sentence formation), we postulated the correct form of the parse tree and converted those rules in if else blocks.
    - ➔ (ROOT (S/(SINV is the correct form, if not, then increment the errors.
    - ➔ (ROOT (SINV (VP is found, then increment the errors.
    - ➔ If FRAG is found, then increment the error
    - ➔ If (NP/(VP is not found before (SBAR (S then increment the error

d) Essay Coherence (Topic coherence, in particular)

i. For pronoun coherence, we extract all the pronoun tags (PRP and PRP$) after pos tagging of the sentence. Then we look at the current sentence (till the word) and previous two sentences to search for the co reference. We extract all the common nouns and proper nouns. We search the common nouns in the wordnet and get its lexname. noun.group, noun.person, noun.body are taken

into consideration and matched with the pronoun cardinality. If it doesn't match, then increase the number of errors. If the POS tag returns a Proper noun, we check the Named Entity of the word using Stanford core NLP Named Entity Recognition. If for singular NER returns anything other than Person we increase the number of errors. Similarly if for plural NER returns anything other than Group we increase error count.

ii. For topic coherence, we extracted the unique topics provided in the csv file and saved an identifying marker in the form of a unique word present in each sentence, in the form of a list. Now, every time the function is called, it checks if the topic has any word corresponding to the unique marker in the list. If such a word exists, a list is called up which contains most probable words, synonyms and antonyms that can occur in a coherent essay and every occurrence of such word is counted. At the end of the run, the function returns a count of occurrences of these words in the essay. It is assumed that higher the word count, more coherent the essay will be.

Patterns of Errors:

- For c i – [NN, VBP], [NNPS, VBZ]
- For c ii – [MD, VB, VB], [VBD, VBN, VBN]
- For c iii – (NN (SBAR (S

After the list of scores were generated for a, b, c.i, c.ii, c.iii, d..i anddii we normalized it using the following equation-

$x = x - \min(x) \;/\; \max(x) - \min(x)$

This gave us values normalized to a range of 0-1, which is then converted to the required scale for each part of the essay.

The final score is generated using the following equation, whose coefficients are different from the original equation, obtained after running linear regression:

0.44412049 * part_a - 0.18825032 * part_b + 0.12061126 * part_c_i - 0.08335527 * part_c_ii + 0.05898674 * part_c_iii + 0.03865994 * part_d_i + 0.05246802 * part_d_ii

Since the problem is nothing but a binary classification now, it became much more simpler and to avoid any use of machine learning which is not really necessary, we simply sorted the list and figured out that the midpoint of final scores was approximately 2.09. So, any score above 2.09 is graded, high and vice versa.

To get the spelling errors in range of 0-4, we multiplied every value with 4.
All the other scores were multiplied by 4 and subtracted from 5 to get the final scores in the range of 1-5. This ensured that lower the number of errors, higher the score obtained by the essay.

All values finally were rounded off to nearest integer value.

Final score was calculated for each essay using the given equation which is:

Final score = 2 * a - b + c.i + c.ii + 2 * c.iii + 2 * d.i + 3 * d.ii

The results are finally written into a text file.