Documentation

Teammate 1 name: Aarsh Patel NetID: 659999805
Teammate 2 name: Raj Mehta NetID: 670579653

How to run:

1. First, make sure you have the following dependencies installed:
    a. nltk
    b. SpaCy
    c. NumPy
    d. Pandas
    e. pyspellchecker
2. Then run the following code in the command line:
   python -m spacy download en_core_web_sm
3. In the "run_project.py" file, before running it, change the path stored in the variable "PATH_TO_ESSAY".
4. If you want to score multiple essays, run a for-loop over the function "*score_essay"*, passing the desired path to essay in each iteration.

Modules and Dependencies:

- spacy: Used for tokenizing text, part-of-speech tagging, and sentence parsing.
- numpy: Provides support for efficient numerical operations on arrays.
- json: Utilized to load and save data in JSON format.
- nltk: Used for text manipulation and pos-tagging.
- pandas: Utilized for handling data in a tabular form.
- pyspellchecker: Used to identify incorrect spelling

sample_code_1.py:

1. general_scorer_gaussian_assumption:

Description: Calculates a score based on a Gaussian (normal) distribution assumption of input data. It converts a raw score (x) into a standardized score within a specified range.

Parameters:

- ***x (float):*** The raw score to be standardized.
- ***mean (float):*** The mean of the distribution.
- ***stddev (float):*** The standard deviation of the distribution.
- ***min_score (int):*** The minimum possible score.
- ***max_score (int):*** The maximum possible score.

- ***reverse (bool, optional):*** If True, reverses the scoring direction.

Returns:

- ***float:*** A score normalized to the specified range and clipped to ensure it remains within the min_score and max_score bounds.

Detailed Behavior:

- Standardization: Converts a raw score x into a z-score using the formula (x - mean) / stddev. Normalization: Maps the z-score to a score within the specified range [min_score, max_score]. It linearly transforms z-scores between -3 and 3 to this range.
- Reversal Option: If reverse is set to True, the direction of scoring is inverted, making higher raw scores correspond to lower normalized scores.
- Clipping: Ensures the final score does not exceed the boundaries set by min_score and max_score.

2. count_sentences_with_spacy:

Description: Counts the number of sentences in a given text using the spacy library.

Parameters:

- ***text (str):*** Text to analyze.

Returns:

- ***int:*** The count of sentences in the text.

Detailed Behavior:

- Text Processing: Uses the spacy library to parse the given text into a document object, which organizes the text into tokens and sentence structures.
- Sentence Extraction: Extracts sentences from the document object and counts them.

3. decontracted:

Description: This function expands English contractions into their full form, which can be helpful for various natural language processing tasks that benefit from standardized text formats.

Parameters:

- ***phrase (str):*** A string containing English text with contractions.

Returns:

- ***str*****:** The input string with all contractions expanded.

Detailed Behavior:

- The function uses regular expressions to replace common English contractions.
- Specific contractions handled include replacements for "won't" to "will not" and "can't" to "can not".

- More general patterns cover other common contractions like "n't" (not), "'re" (are), "'s" (is), etc.

4. num_sentences:

Description: Evaluates the text based on the number of sentences, using a scoring system based on a Gaussian distribution of known sentence counts.

Parameters:

- ***text (str):*** Text to evaluate.
- ***sentence_counts (list):*** Historical data of sentence counts.
- ***min_score (int):*** Minimum score to assign.
- ***max_score (int):*** Maximum score to assign.

Returns:

- ***float*:** A score reflecting the appropriateness of the sentence count in the text.

Detailed Behavior:

- Initial Check: Directly returns min_score if the sentence count is 10 or less.
- Data Filtering: Filters out sentence counts from historical data that are 10 or less (non-informative data).
- Statistical Analysis: Calculates the mean and standard deviation of the filtered historical sentence counts.
- Scoring: Applies the general_scorer_gaussian_assumption to the counted sentences, scoring them based on how they compare statistically to historical data.

5. spell_check:

Description: This function checks the spelling of words in a text and calculates the percentage of misspelled words. It first expands contractions using the decontracted function, then tokenizes the text, lemmatizes each word, and finally checks for spelling errors.

Parameters:

- ***text (str):*** A string of text in which to check spelling.

Returns:

- ***float*:** The percentage of misspelled words in the text, rounded to two decimal places.

Detailed Behavior:

- The text is first decontracted to normalize contractions.
- nltk.word_tokenize is used for tokenizing the string into words.
- Each word is lemmatized using nltk.stem.WordNetLemmatizer to reduce it to its base or dictionary form.
- spellchecker.SpellChecker is utilized to identify words not recognized by its dictionary.

- The function calculates the percentage of words identified as misspelled compared to the total number of words.

6. spelling_mistakes:

Description: Evaluates the text based on the percentage of spelling mistakes, using a scoring system based on a Gaussian distribution of known spelling mistake rates.

Parameters:
- ***text (str):*** Text to evaluate.
- ***mistakes_list (list):*** Historical data of spelling mistakes percentages.
- ***min_score (int):*** Minimum score to assign.
- ***max_score (int):*** Maximum score to assign.

Returns:
- ***float*****:** A score reflecting the quality of spelling in the text.

Detailed Behavior:
- Input Handling: Accepts text, minimum score, and maximum score as parameters.
- Text Preparation: Expands contractions, tokenizes and lemmatizes the text, and identifies misspelled words using SpellChecker.
- Error Quantification: Calculates the percentage of misspelled words in the text.
- Score Calculation: Compares this percentage against a known distribution of errors, converting it to a score within the specified range using a Gaussian distribution model.
- Output: Outputs a score where higher values represent better spelling quality relative to typical levels.

sample_code_2.py:

1. general_scorer_gaussian_assumption:

Description: Calculates a score based on a Gaussian (normal) distribution assumption of input data. It converts a raw score (x) into a standardized score within a specified range.

Parameters:
- ***x (float):*** The raw score to be standardized.
- ***mean (float):*** The mean of the distribution.
- ***stddev (float):*** The standard deviation of the distribution.
- ***min_score (int):*** The minimum possible score.
- ***max_score (int):*** The maximum possible score.
- ***reverse (bool, optional):*** If True, reverses the scoring direction.

Returns:

- ***float**:* A score normalized to the specified range and clipped to ensure it remains within the min_score and max_score bounds.

Detailed Behavior:

- Standardization: Converts a raw score x into a z-score using the formula (x - mean) / stddev. Normalization: Maps the z-score to a score within the specified range [min_score, max_score]. It linearly transforms z-scores between -3 and 3 to this range.
- Reversal Option: If reverse is set to True, the direction of scoring is inverted, making higher raw scores correspond to lower normalized scores.
- Clipping: Ensures the final score does not exceed the boundaries set by min_score and max_score.

2. agreement:

Description: Evaluates subject-verb agreement in a given text and assigns a score based on the frequency of errors.

Parameters:

- ***text (str):*** The text to analyze.
- ***min_score (float):*** The minimum score possible.
- ***max_score (float):*** The maximum score possible.

Returns:

- ***score (float):*** The score indicating the quality of subject-verb agreement.

Detailed Behavior:

- Uses **spacy** to parse the text and identify subjects and verbs.
- Counts the instances of subject-verb disagreement based on predefined rules.
- Scores the fraction of errors using a Gaussian assumption against historical data.

.

3. verbs:

Description: Analyzes verb tense consistency within sentences and scores the text based on the prevalence of unlikely verb tense changes.

Parameters:

- **text (str):** The text to analyze.
- **min_score (float):** The minimum score possible.
- **max_score (float):** The maximum score possible.

Returns:

- ***score (float):*** The score indicating the quality of verb tense usage.

Detailed Behavior:

- Tokenizes the text into sentences and then into words.

- Tags the words with part-of-speech tags.
- Evaluates the probability of sequential verb tags and identifies low-probability transitions as mistakes.
- Scores the percentage of mistakes against a Gaussian distribution using historical error data.

4. subject_verb_disagree:

Description: Determines if there is a disagreement between a subject and its corresponding verb based on predefined grammatical rules.

Parameters:

- **subject (spacy.tokens.Token):** The subject token from spaCy's parsing.
- **verb (spacy.tokens.Token):** The verb token from spaCy's parsing.

Returns:

- ***bool**:* Returns **True** if there is a disagreement between the subject and verb, otherwise **False**.

Detailed Behavior:

- Checks if the subject's part-of-speech tag is in the predefined list of disagreement rules (subject_verb_disagreements).
- If the verb's tag is also in the list associated with the subject's tag, it returns True, indicating a disagreement. Otherwise, it returns False.

5. count_subject_verb_errors_fraction:

Description: Calculates the fraction of subject-verb disagreements within the text.

Parameters:

- ***text (str):*** The text to analyze for subject-verb agreement errors.

Returns:

- ***float:*** The fraction of tokens in the text that are involved in subject-verb disagreements.

Detailed Behavior:

- Parses the text with spaCy to tokenize it and identify sentences.
- Iterates through each sentence and token, searching for verbs.
- For each verb, it checks if there is a subject connected to it and evaluates their agreement using **subject_verb_disagree**.
- Calculates the fraction of total tokens that are involved in disagreements.

6. tag_probability:

Description: Calculates the probability of observing a specific part-of-speech tag following another, based on a conditional frequency distribution from the Brown corpus.

Parameters:

- ***prev_tag (str):*** The previous word's part-of-speech tag.
- ***current_tag (str*)**:** The current word's part-of-speech tag.

Returns:

float: The probability of current_tag following prev_tag.

Detailed Behavior:

- Loads a previously saved conditional frequency distribution (**cfd**) object.
- Computes the frequency of **current_tag** following **prev_tag** and the total frequencies of all tags following **prev_tag**.
- Returns the conditional probability of **current_tag** given **prev_tag**.

7. verb_mistakes:

Description: Evaluates verb tense consistency by analyzing the sequence of verb tenses in a text and identifying unlikely transitions.

Parameters:

text (str): The text to analyze for verb tense mistakes.

Returns:

float: The percentage of verb tense mistakes relative to the total number of words.

Detailed Behavior:

- Tokenizes the text into sentences and then words, and tags each word with its part-of-speech.
- Iterates over consecutive tags, focusing on verbs.
- Uses **tag_probability** to compute the probability of each verb tag transition.
- Counts transitions with a probability lower than 0.05 as mistakes.
- Returns the percentage of total tags that are considered mistakes based on these transitions.

run_project.py:

1. score_essay:

Description: Calculates a composite score for an essay based on multiple linguistic metrics, providing a holistic evaluation of its quality.

Parameters:

- ***PATH_TO_ESSAY (str):*** The file path to the essay text file.

Returns:

- ***sents_score*** *(float):* The calculated sentence score of the essay.
- ***spell_score*** *(float):* The calculated spelling score of the essay.
- ***agree_score*** *(float):* The calculated agreement score of the essay.
- ***verbs_score*** *(float):* The calculated verb tense score of the essay.
- ***final_score*** *(float):* The calculated final score of the essay.

Detailed Behavior:

- **Reading the Essay**: The function opens and reads the full text of the essay from the specified file path
- **Sentence Count Evaluation**: It scores the essay based on the number of sentences using the **num_sentences** function.
- **Spelling Mistakes Evaluation**: It evaluates spelling mistakes using the **spelling_mistakes** function and scores the essay accordingly.
- **Subject-Verb Agreement Evaluation**: It assesses subject-verb agreement with the **agreement** function and calculates a score.
- **Verb Tense Consistency Evaluation**: It examines verb tense consistency using the **verbs** function and generates a score.
- **Score Calculation**: Scores from each metric are combined using a formula that doubles the sentence score, subtracts the spelling score, and adds the agreement and verb scores.