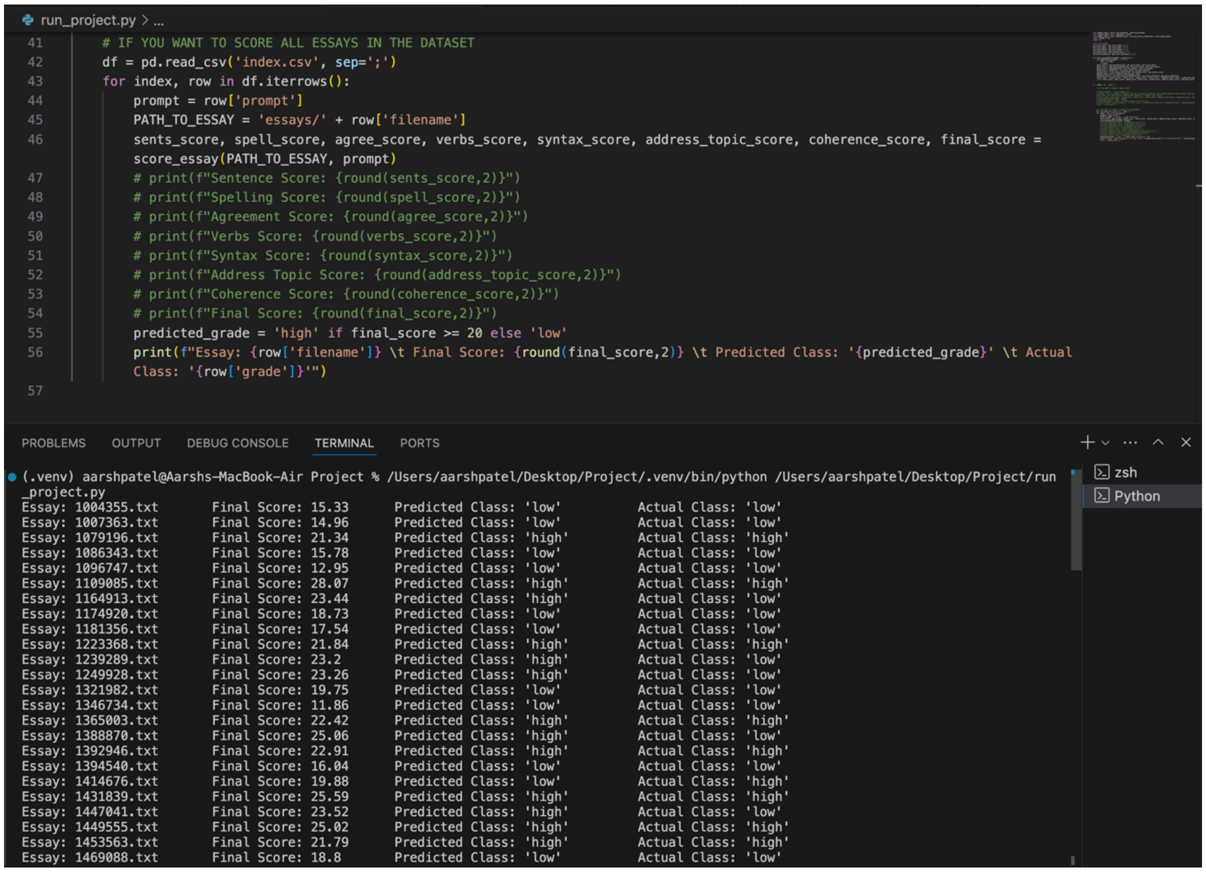
Documentation

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
   f. Scikit-learn
   g. Matplotlib
2. Then run the following code in the command line:
   python -m spacy download en_core_web_sm
3. **To score a single essay**, uncomment line 34 to 38 in "run_project.py" file, change the path stored in the variable "PATH_TO_ESSAY" and the "prompt"
4. To score all essays in the Dataset, simply run the "run_project.py" file without making any changes

Sample Output:



```python
# IF YOU WANT TO SCORE ALL ESSAYS IN THE DATASET
df = pd.read_csv('index.csv', sep=';')
for index, row in df.iterrows():
    prompt = row['prompt']
    PATH_TO_ESSAY = 'essays/' + row['filename']
    sents_score, spell_score, agree_score, verbs_score, syntax_score, address_topic_score, coherence_score, final_score =
    score_essay(PATH_TO_ESSAY, prompt)
    # print(f"Sentence Score: {round(sents_score,2)}")
    # print(f"Spelling Score: {round(spell_score,2)}")
    # print(f"Agreement Score: {round(agree_score,2)}")
    # print(f"Verbs Score: {round(verbs_score,2)}")
    # print(f"Syntax Score: {round(syntax_score,2)}")
    # print(f"Address Topic Score: {round(address_topic_score,2)}")
    # print(f"Coherence Score: {round(coherence_score,2)}")
    # print(f"Final Score: {round(final_score,2)}")
    predicted_grade = 'high' if final_score >= 20 else 'low'
    print(f"Essay: {row['filename']} \t Final Score: {round(final_score,2)} \t Predicted Class: '{predicted_grade}' \t Actual
    Class: '{row['grade']}'")
```

```
(.venv) aarshpatel@Aarshs-MacBook-Air Project % /Users/aarshpatel/Desktop/Project/.venv/bin/python /Users/aarshpatel/Desktop/Project/run
_project.py
Essay: 1004355.txt       Final Score: 15.33      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1007363.txt       Final Score: 14.96      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1079196.txt       Final Score: 21.34      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1086343.txt       Final Score: 15.78      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1096747.txt       Final Score: 12.95      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1109085.txt       Final Score: 28.07      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1164913.txt       Final Score: 23.44      Predicted Class: 'high'         Actual Class: 'low'
Essay: 1174920.txt       Final Score: 18.73      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1181356.txt       Final Score: 17.54      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1223368.txt       Final Score: 21.84      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1239289.txt       Final Score: 23.2       Predicted Class: 'high'         Actual Class: 'low'
Essay: 1249928.txt       Final Score: 23.26      Predicted Class: 'high'         Actual Class: 'low'
Essay: 1321982.txt       Final Score: 19.75      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1346734.txt       Final Score: 11.86      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1365003.txt       Final Score: 22.42      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1388870.txt       Final Score: 25.06      Predicted Class: 'high'         Actual Class: 'low'
Essay: 1392946.txt       Final Score: 22.91      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1394540.txt       Final Score: 16.04      Predicted Class: 'low'          Actual Class: 'low'
Essay: 1414676.txt       Final Score: 19.88      Predicted Class: 'low'          Actual Class: 'high'
Essay: 1431839.txt       Final Score: 25.59      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1447041.txt       Final Score: 23.52      Predicted Class: 'high'         Actual Class: 'low'
Essay: 1449555.txt       Final Score: 25.02      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1453563.txt       Final Score: 21.79      Predicted Class: 'high'         Actual Class: 'high'
Essay: 1469088.txt       Final Score: 18.8       Predicted Class: 'low'          Actual Class: 'low'
```

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
- scikit-learn: Used for model training and sentence verctorization
- matplotlib: Used for visualizations

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.

sample_code_3.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. get_topical_part:

Description: Extracts and returns the second sentence from the input prompt if it contains three or more sentences. If the prompt contains fewer than three sentences, it returns the entire prompt.

Parameters:

- ***prompt (str):*** The text input from which to extract the topical part.

Returns:
- ***str:*** The second sentence of the prompt if there are at least three sentences, otherwise the entire prompt.

Detailed Behavior:
- Tokenizes the input text into sentences using the nltk.sent_tokenize function.
- Checks the number of sentences in the prompt.
- If the prompt contains three or more sentences, it returns the second sentence as the topical part.
- If there are fewer than three sentences, returns the entire prompt.

3. get_similarity_score:

Description: Computes a similarity score between two text inputs using TF-IDF vectorization to quantify their semantic similarity. The function adjusts the text by removing common English stopwords before scoring.

Parameters:
- ***text1 (str):*** The first text input.
- ***text2 (str):*** The second text input.

Returns:
- ***score (int):*** A similarity score on a scale from 1 to 5, where 1 indicates very low similarity and 5 indicates very high similarity.

Detailed Behavior:
- Stopword Removal: Both text inputs are stripped of common English stopwords to focus on more meaningful words.
- TF-IDF Vectorization: Transforms the adjusted texts into TF-IDF vectors. This method weighs the texts' terms based on their importance, which is determined by how frequently they appear in the text relative to their frequency in a corpus.
- Similarity Calculation: Computes the cosine similarity between the two TF-IDF vectors. This similarity is a numerical value between 0 and 1, where 0 means no similarity and 1 means complete similarity.
- Scoring: The similarity value is then translated into an integer score from 1 to 5 based on predefined thresholds.
- The scores are assigned as follows:
  - Less than 0.1 similarity scores 1.
  - 0.1 to less than 0.2 scores 2.
  - 0.2 to less than 0.4 scores 3.
  - 0.4 to less than 0.6 scores 4.
  - 0.6 and above scores 5.

4. address_topic:

Description: This function aims to assess the relevance of an essay to the topic specified in the prompt. It first extracts the key topic from the prompt, then compares this topic with the content of the essay to determine their similarity.

Parameters:
- ***prompt (str):*** The prompt containing the topic that the essay should address.
- ***essay (str):*** The essay text that is to be evaluated.

Returns:

- ***float:*** A score representing the degree of similarity between the topic of the prompt and the content of the essay, indicating how well the essay addresses the prompt.

Detailed Behavior:

- Extracts the topical part of the prompt using `get_topical_part`, which isolates the main subject or theme from the prompt.
- Computes a similarity score between the extracted topic and the essay content using `get_similarity_score`. This function uses text comparison techniques to quantify how closely the essay content aligns with the topic described in the prompt.
- Returns the similarity score, which quantifies the relevancy of the essay to the prompt. Higher scores indicate a closer alignment between the essay's content and the prompt's topic.

5. traverse_tree:

Description: This function recursively traverses a tree structure, specifically designed for processing natural language syntax trees. It builds a hierarchical dictionary that maps parents to their children and grandchildren, capturing the labels of subtrees (nodes) and terminal nodes (leaves).

Parameters:

- ***tree (nltk.tree.Tree):*** The tree to be traversed, typically parsed from natural language data using nltk's parsing capabilities.

Returns:

- ***None:*** This function modifies the global variable `parent_child_grandchild` in-place, and does not return any value**.**

Detailed Behavior:

- Iteration: Iterates through each subtree within the given tree.
- Parent-Child-Grandchild Mapping: For each node in the tree, checks if it is a subtree (nltk.tree.Tree type). If so, it captures this node as a 'parent'. It then checks for children within this subtree. If the children are also subtrees, it considers them as 'child' nodes and further checks for 'grandchild' nodes within.
- Dictionary Update: Updates the global dictionary `parent_child_grandchild` where each key is a parent node label, and the value is another dictionary mapping child labels to a list of grandchild labels or terminal node values.
- Recursion: The function is called recursively on each subtree to process all levels of the tree structure.
- Error Handling: The function assumes the presence of certain structure and types, and may not handle unexpected formats or data types gracefully.

6. check_sentence_syntax:

Description: Analyzes the syntax of a given sentence by parsing its structure using a predefined grammar and checking against a hierarchy of part-of-speech tags defined in checker_dict. It counts the instances where the structure does not comply with the expected tag relationships.

Parameters:

- ***sentence (list of str):*** The sentence to analyze, provided as a list of words.
- ***checker_dict (dict):*** A dictionary defining valid parent-child relationships between part-of-speech tags. The dictionary is structured such that the keys are parent tags, and the values are dictionaries where keys are child tags and values are lists of permissible sub-child tags.

Returns:

- ***int:*** The count of syntactical mistakes found in the sentence based on the provided tag hierarchy.

Detailed Behavior:

- The function first tokenizes the sentence into words and tags them using NLTK's part-of-speech tagger.
- It then parses these tagged tokens using an NLTK RegexpParser with a predefined grammar.
- The parser output, a tree structure, is traversed. For each node (tagged token) in this tree:
    - The function checks if the node's tag (parent) is in the checker_dict.
    - If it is, it further checks if the immediate child's tag is a valid child according to checker_dict.
    - It then checks for the validity of the sub-child's tag under the child's tag in the checker_dict.
    - If any of these checks fail (either the parent, child, or sub-child tag is not valid as per the checker_dict), increments the mistake count.
- Finally, the function returns the total count of these mistakes.

7. score_essay_syntax:

Description: Scores an essay based on its syntax errors. The function calculates the number of mistakes across all sentences in the essay, then uses a Gaussian distribution assumption to compute a standardized score within a specified range.

Parameters:

- ***text (str):*** The text of the essay to be analyzed.
- ***min_score (float):*** The minimum possible score.
- ***max_score (float):*** The maximum possible score.

Returns:

- ***float:*** A score that represents the syntactic quality of the essay, normalized to the specified range.

Detailed Behavior:

- Tokenization: Splits the essay into sentences and then tokenizes each sentence into words.
- Error Detection: Uses a predefined dictionary (checker_dict) to check each tokenized sentence for syntax errors, counting the total mistakes.
- Standardization and Normalization: Computes the mean and standard deviation of mistakes across a dataset (presumably stored in mistakes_per_essay). It then uses these statistics to convert the mistake count into a standardized score, considering higher mistakes as worse, hence the use of the 'reverse' parameter set to True in the general_scorer_gaussian_assumption function.
- Clipping: Ensures the final score is clipped to remain within the boundaries defined by min_score and max_score.

8. get_all_sent_embeddings:

Description: Extracts sentence embeddings for each sentence in a given text using a pre-defined NLP model. This function processes the entire essay, breaking it down into individual sentences, and computes their embeddings.

Parameters:

- ***essay (str):*** The text of the essay from which to extract sentence embeddings.

Returns:

- ***list:*** A list of embeddings, where each embedding corresponds to a sentence in the essay.

Detailed Behavior:

- Initializes an NLP model (assumed to be pre-loaded and referenced as `nlp`) to process the input text.
- Parses the essay into sentences using the NLP model's sentence segmentation capability.
- For each sentence, calculates its embedding using a function `sent_embedding`, which must be predefined or imported elsewhere in the code.
- Compiles the embeddings of all sentences into a list and returns this list.

9. get_pairwise_cosine_similarities:

Description: Computes the pairwise cosine similarities between consecutive sentence embeddings in a given text (essay).

Parameters:

- ***essay (str):*** The text of the essay from which sentences will be extracted and their embeddings calculated.

Returns:

- ***list of float:*** A list containing the cosine similarity scores between each consecutive pair of sentence embeddings.

Detailed Behavior:

- Extracts sentence embeddings from the input essay using the `get_all_sent_embeddings` function.
- Iterates through the list of embeddings, computing the cosine similarity between each pair of consecutive embeddings.
- Stores each similarity score in a list and returns this list.

10. compute_coherence_changes:

Description: Computes the coherence changes within an essay based on pairwise cosine similarities between sentences. The function calculates the number of significant changes in similarity and the cumulative magnitude of these changes.

Parameters:

- ***essay (str***): The text of the essay to analyze.
- ***threshold (float):*** A threshold value that defines what constitutes a significant change in similarity. Changes are considered significant if the relative change in similarity exceeds this threshold.

Returns:

- ***tuple:***
  - ***change_value (float):*** The cumulative magnitude of significant changes in similarity. This is the sum of the absolute relative changes that exceed the threshold.
  - ***changes (int):*** The number of times significant changes in similarity occur between consecutive sentences.

Detailed Behavior:

- The function first computes the pairwise cosine similarities for consecutive sentences in the essay using a helper function get_pairwise_cosine_similarities.

- It then iterates through these similarity scores. For each pair of consecutive scores, it checks if the relative change in similarity exceeds the defined threshold.
- A relative change is calculated as the absolute value of the difference between two consecutive similarities divided by the first similarity value.
- If the relative change exceeds the threshold, it increments a counter for the number of changes and adds the relative change to a cumulative change value.
- The function finally returns the total change value and the count of significant changes as a tuple.
- If no sentences are detected or similarities can't be calculated (e.g., an empty essay), the function returns (0, 0).

11. score_by_essay_incoherence:

Description: Scores an essay based on its incoherence level. The function first computes incoherence changes within the text and then utilizes a Gaussian scoring model to assign a standardized score. The score is designed to inversely correlate with the incoherence level: more coherent essays receive higher scores.

Parameters:
- ***text (str):*** The essay text to be analyzed for incoherence.
- ***min_score (float):*** The minimum possible score that can be assigned.
- ***max_score (float):*** The maximum possible score that can be assigned.

Returns:
- ***float:*** The score reflecting the incoherence level of the essay, adjusted to the specified scoring range.

Detailed Behavior:
- **Coherence Computation:** The function calculates changes in coherence through the **compute_coherence_changes** method, which quantifies the shifts in coherence relative to a threshold of 0.1.
- **Initial Scoring:** If the essay exhibits one or fewer coherence changes, it immediately receives the maximum score, reflecting high coherence.
- **Score Calculation with Gaussian Assumption:**
  - The mean and standard deviation are calculated from a dataset (**incoherence_in_essays**), representing typical incoherence levels in essays.
  - Utilizes the **general_scorer_gaussian_assumption** function, providing it with the computed incoherence level, mean, standard deviation, the specified scoring range, and a reversal flag set to True. The reversal ensures that higher incoherence levels translate to lower scores.
- **Score Clipping:** The Gaussian scoring function also ensures that the score does not fall outside the provided minimum and maximum bounds, regardless of the input incoherence level.

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.
- ***prompt (str, optional):*** The essay prompt against which the relevance of the essay's content is assessed.

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.
- ***syntax_score*** *(float):* The score reflecting the syntactical structure of the essay.
- ***address_topic_score*** *(float):* The score evaluating how well the essay addresses the given prompt.
- ***coherence_score*** *(float):* The score assessing the coherence of the essay.
- ***final_score*** *(float):* The calculated final score of the essay.
- ***Predicted_grade*** *(str):* The predicted grade of 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.
- **Syntax Evaluation:** Scores essay syntax using the **score_essay_syntax** function.
- **Topic Addressing Evaluation:** Assesses how well the essay addresses the provided prompt using the **address_topic** function, if a prompt is given.
- **Coherence Evaluation:** Evaluates essay coherence with the **score_by_essay_incoherence** function.
- **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.
- **Grade Prediction**: If the finale score is <=20 then it assigns "low" grade otherwise it assigns "high" grade.

grad_student_part_2.ipynb:

Overview

This documentation outlines the implementation of three classifiers—Naive Bayes, Logistic Regression, and Multi-layer Perceptron—to predict the scoring category (high or low) of essays based on various linguistic features. The classifiers use metrics computed from the essay texts to train and evaluate their predictions.

Data Preparation

- **Input Data**: Read from **index.csv**, which includes filenames and associated prompts.
- **Feature Extraction**: Each essay is processed to compute features such as sentence counts, spelling mistakes, grammatical agreement, verb tense consistency, syntax, topic relevance, and coherence.

Training and Test Sets

- **Splitting Data**: The dataset is randomly split into training (80%) and testing (20%) sets using a fixed random seed for reproducibility.
- **Feature Labels**: Extracted features are used as inputs (X), and the binary target score (high/low encoded as 1/0) is used as the output (y).

Classifiers

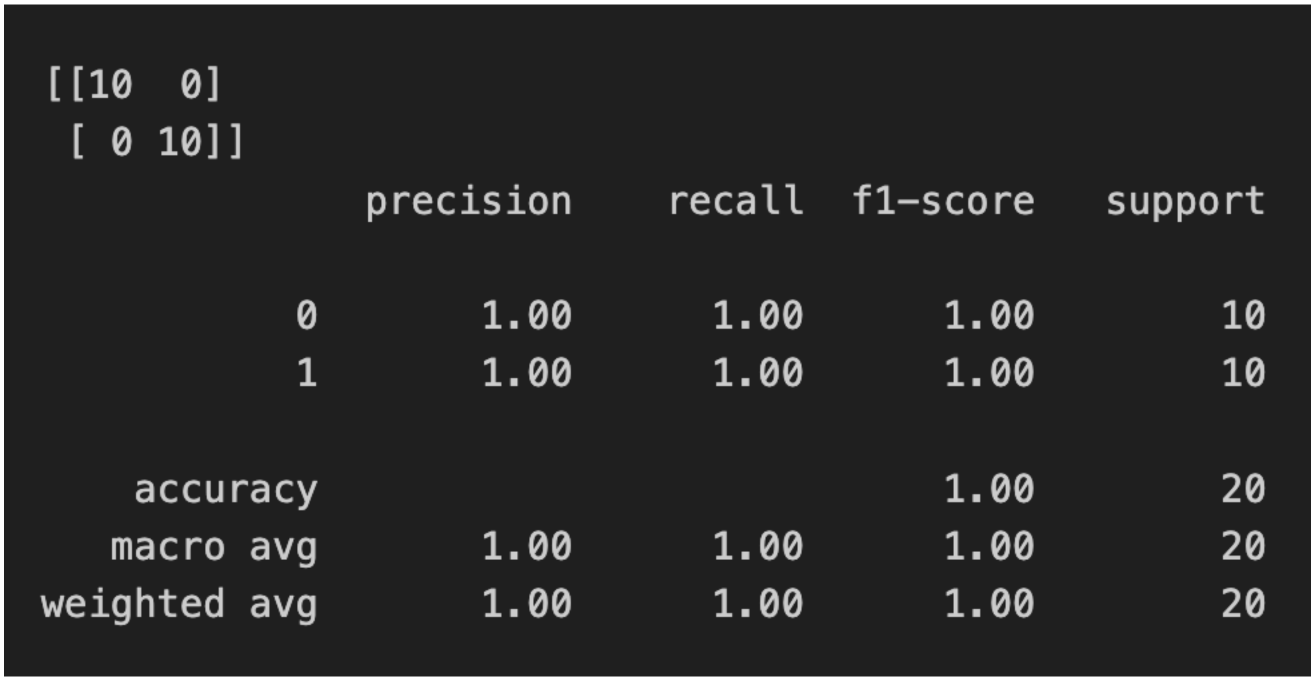
1. **Naive Bayes (GaussianNB)**
2. **Logistic Regression**

3. **Multi-layer Perceptron (MLPClassifier)**

Metrics and Visualization

- **Accuracy**: Reports the percentage of correctly predicted instances.
- **Precision-Recall Curve**: Plots to evaluate the trade-off between precision and recall for the positive class.
- **AUC-ROC Curve**: Shows the performance across all classification thresholds.
- **Confusion Matrix and Classification Report**: Provides detailed insights into the true positives, false positives, true negatives, and false negatives.

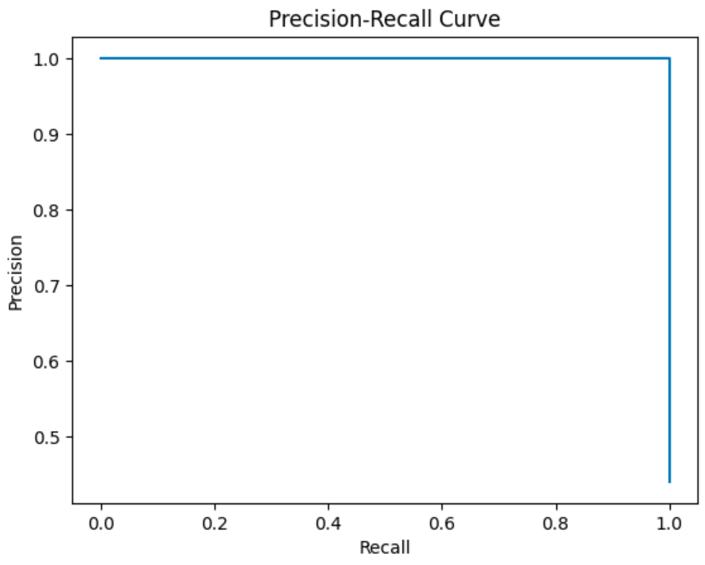
Sample Output

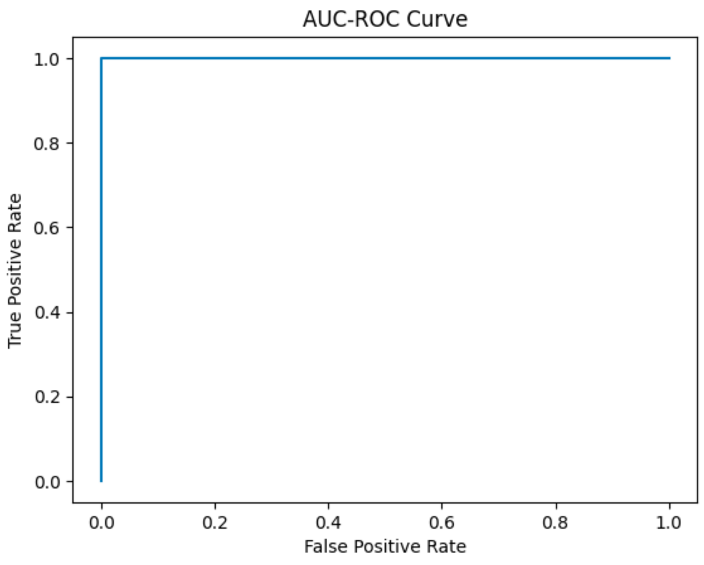


```
[[10  0]
 [ 0 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```


Precision-Recall Curve
Precision
Recall


AUC-ROC Curve
True Positive Rate
False Positive Rate