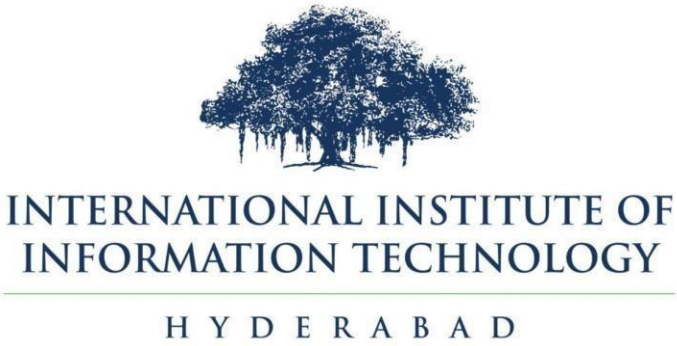
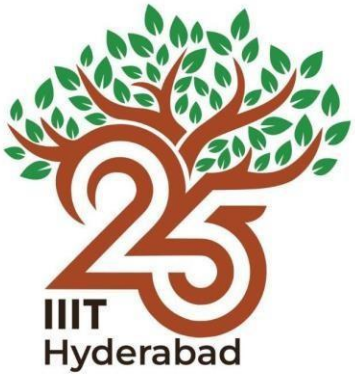




AUTOMATED ESSAY SCORING

September 2023

COURSE CODE: INFORMATION RETRIEVAL & EXTRACTION – CS4.406.M23

Advisor:
Prof. Rahul Mishra

Mentor:
Gokul Vamsi Thota

Team Number:
1

Representatives:
Vaibhav Mittal - 2022201016
Varun Vashishtha - 2022201061
Tarun Kumar - 2022201008

Academic year:
2023-2024

Contents

1. Project Oveview:

Problem Statement:In AES, the input to the system is a student essay, and the output is the score assigned to the essay. The score assigned by the AES system will be compared against the human assigned score to measure their agreement. Common agreement measures used include Pearson's correlation, Spearman's correlation, and quadratic weighted Kappa (QWK).

Understanding the Problem Statement:The problem entails addressing several key aspects:

- ☐ **AES System Input**: The input to the AES system is a student essay. This is the text that the system will analyze and evaluate.
- ☐ **AES System Output**: The output of the AES system is the score assigned to the essay. This score is a numerical representation of the quality or performance of the essay as determined by the system.
- ☐ **Comparison with Human Scores**: The AES system's assigned score is going to be compared against scores assigned by humans. This is done to assess how well the AES system's evaluations align with human judgment, which is often considered the gold standard in essay grading.
- ☐ **Agreement Measures**: To measure the agreement between the scores given by the AES system and the human-assigned scores, several common agreement measures are mentioned. These include:
    - Pearson's Correlation: A statistical measure of the linear relationship between two sets of data. It assesses the strength and direction of the linear association.
    - Spearman's Correlation: A non-parametric measure of the strength and direction of association between two ranked variables. It is based on the ranks of the data, making it suitable for ordinal data.
    - Quadratic Weighted Kappa (QWK): A measure of agreement between two raters, considering the possibility of agreement occurring by chance. It is often used in situations where multiple raters are assessing the same item.

2. Data Overview:

The most widely used dataset for AES is the Automated Student Assessment Prize (ASAP) , and it has been utilized to evaluate the performance of AES systems . Generally, ASAP is composed of 8 prompts and 12976 essays, which are written by students from Grade 7 to Grade 10.

DETAILS ABOUT ASAP DATASET

PROMPT ID	ESSAYS	SCORE RANGE
1	1783	2-12
2	1800	1-6
3	1726	0-3
4	1772	0-3
5	1805	0-4
6	1800	0-4
7	1569	0-30
8	723	0-60

In the ASAP dataset, score ranges are different from each other. For consistency, the scores in the training set are first normalized to a particular range . During the testing process, predicted scores will be rescaled to the original range.

3. Approach followed:

We have implemented two types of AES models, i.e., **feature-engineered models** and **end-to-end models**

The first type, feature-engineered models, rely on manually designed features like **word count** and **grammar errors**. These models have the advantage of being explainable and adaptable to various scoring criteria. However, they often **struggle to capture deep semantic insights**, especially for essays that depend on understanding specific prompts.

The second category of Automated Essay Scoring (AES) models is the **end-to-end model**, influenced by advancements in deep learning. These models transform essays into low-dimensional vectors using **word embedding**, then use a dense layer to convert these vectors with deep **semantic meanings** into ratings.

Our implementation combines both feature engineered models and end-to-end models, sentence embeddings are derived by the **pre-trained BERT model**.

Preprocessing of data:

STEP 1:
In our code, we define a series of functions to process text, including making it lowercase, removing punctuation (excluding periods and adding "@"), and removing extra gaps, with the pipeline function applying these steps to a given input text.

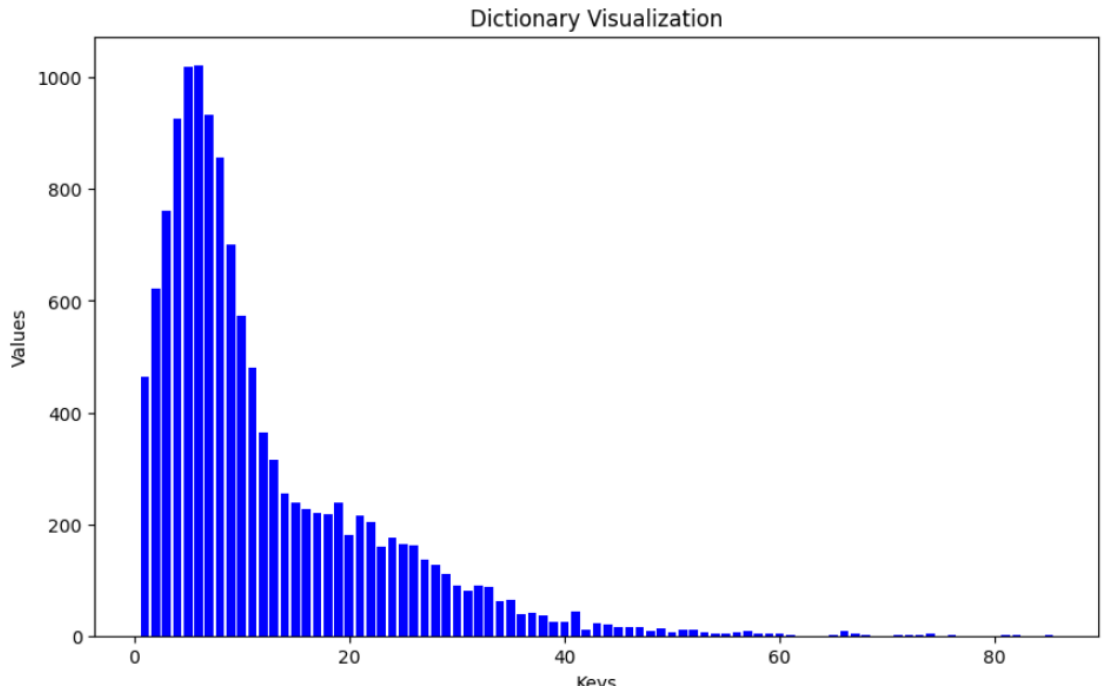
```
string.punctuation
punct = string.punctuation
punct = punct.replace('.', '')
punct += '@'
'.' in punct

def get_lower(text):
    return text.lower()

def remove_punctuations(text):
    return ''.join([char for char in text if char not in punct])

def tokenize(text):
    # text = text.strip()
    return word_tokenize(text)

def remove_alpha_numeric(sentence):
    # return ' '.join(word for word in tokens if word.isalpha())
    words = sentence.split()
    alphabetic_words = [word for word in words if word.isalpha()]
    return ' '.join(alphabetic_words)

def remove_tags(text):
    return BeautifulSoup(text, 'html.parser').get_text()

def remove_extra_gaps(text):
    return ' '.join(text.split())

def pipeline(text):
    text = get_lower(text)
    text = remove_punctuations(text)
    # tokens = tokenize(text)
    # text = remove_alpha_numeric(text)
    # text = remove_tags(text)
    text = remove_extra_gaps(text)
    return text
```

STEP 2:

```
ess = df['essay']
maxi = 0
d = {}
for e in ess:
  size = len(sent_tokenize(e))
  if size in d:
    d[size] += 1
  else:
    d[size] = 1
```

```
import matplotlib.pyplot as plt

keys = list(d.keys())
values = list(d.values())

plt.figure(figsize=(10, 6))

# Create bar chart
plt.bar(keys, values, color='blue')

# Add title and labels
plt.title('Dictionary Visualization')
plt.xlabel('Keys')
plt.ylabel('Values')

plt.show()
```

In this code, we count the number of sentences in each essay stored in the 'essay' column of a DataFrame, storing the counts in a dictionary with the number of sentences as keys and the count of essays with that number of sentences as values.We then visualize this dictionary using histogram.



STEP 3:
Preprocessing of text using BERT:

```
example_text = 'I will watch Memento tonight. Do you'
bert_input = tokenizer(example_text,padding='max_length', max_length = 10,
                       truncation=True, return_tensors="pt")


print(bert_input['input_ids'])
print(bert_input['token_type_ids'])
print(bert_input['attention_mask'])
tensor([[  101,   146,  1209,  2824,  2508, 26173,  3568,   102,     0,     0]])
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 0, 0]])
```

```
tensor([[  101,  1045,  2097,  3422,  2033, 23065,  3892,  1012,  2079,   102]])
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

tensor([[1, 1, 1, 1, 1, 1, 1, 1, 0, 0]])
```

- bert_input['input_ids']: This tensor represents the input text converted into token IDs, including special tokens (101 for [CLS], 102 for [SEP], and 0 for padding).
- bert_input['token_type_ids']: This tensor indicates the segment or token type of each token, but in this case, it's all zeros, meaning the input is treated as a single segment.
- bert_input['attention_mask']: This tensor is used to indicate which tokens the model should pay attention to (1 for actual tokens, 0 for padding tokens).

In the output, you can see the tokenized and padded input text and their corresponding token IDs, token type IDs, and attention mask.

STEP 4:
Normalization of domain score:In the below code, we normalize the 'domain_score' values for

essays within different sets (ranging from 1 to 8) in a DataFrame. It does this by linearly scaling the scores within each set to a new range of 0 to 10.

```
for essay_set in range(1, 9):  # Change the range to cover all 8 essay sets
    temp = temp_df[temp_df['essay_set'] == essay_set]
    min_value = temp['domain1_score'].min()
    max_value = temp['domain1_score'].max()
    print(min_value, max_value)
    temp_df.loc[temp_df['essay_set'] == essay_set, 'normalized_score'] = (temp['domain1_score'] - min_value) / (max
```

Then we analyzed normalized scores by looping through each essay set from 1 to 8 and calculating the minimum and maximum values of the 'normalized_score' within the temporary DataFrame temp for the current essay set.

STEP 5:
Extracting sentence level embeddings from the pretrained-BERT model:
In the below code, we showcase how to use a pre-trained BERT model to generate sentence embeddings from text data, which is used in subsequent tasks.

```
import torch
import numpy as np
from nltk.tokenize import sent_tokenize  # We'll use NLTK to tokenize the essay into sentences
from transformers import BertTokenizer, BertModel  # Ensure you've imported these

class BertSentenceEmbedding(nn.Module):
    def __init__(self, pretrained_model_name='bert-base-uncased'):
        super(BertSentenceEmbedding, self).__init__()  # Call the super class's __init__ first

        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.tokenizer = BertTokenizer.from_pretrained(pretrained_model_name)
        self.model = BertModel.from_pretrained(pretrained_model_name, output_hidden_states=True)
        self.model.eval()

    def get_embedding(self, text):

        # Tokenize the essay into sentences
        sentences = sent_tokenize(text)

        # List to hold embeddings for each sentence
        sentence_embeddings = []

        for sentence in sentences:
            # I noticed you were using a "pipeline" function that was not defined in the given code
            processed_sentence = pipeline(sentence)
            inputs = self.tokenizer(sentence, return_tensors="pt", truncation=True, padding=True, max_length=512)
            with torch.no_grad():
                outputs = self.model(**inputs.to(self.device))

            # Use the penultimate layer's hidden states
            hidden_states = outputs.hidden_states[-2]

            # Compute the mean of all tokens embeddings for this sentence
            sentence_embedding = torch.mean(hidden_states, dim=1).squeeze().cpu().numpy()
            sentence_embeddings.append(sentence_embedding)

        return np.array(sentence_embeddings)  # Return embeddings for all sentences in the essay

# Assuming you've defined device somewhere above
bert_embedder = BertSentenceEmbedding().to(device)
essay = df['essay'][0]
embeddings = bert_embedder.get_embedding(essay)
```

The get_embedding method takes an input text, tokenizes it into sentences using NLTK's sent_tokenize, and then processes each sentence separately.

- Each sentence is passed through the BERT tokenizer and model, with settings for truncation, padding, and a maximum length of 512 tokens.
- The penultimate layer's hidden states (from outputs.hidden_states[-2]) are extracted.
- The mean of all token embeddings for each sentence is computed, resulting in a sentence-level embedding.
- These sentence embeddings are collected in a list.

Finally,we return the NumPy array of these embeddings.

FIRST STAGE:

Semantic Score: In the first stage, we utilize LSTM/BI-LSTM to map essays into low-dimensional embeddings, which are then fed to a dense output layer for scoring essays.It is prompt-independent, and utilized to evaluate essays from a deep semantic level.

The SemanticScore class is designed to calculate semantic scores for input text data. It initializes with hyperparameters like the dimension of BERT embeddings, LSTM settings, and dropout probabilities. In the forward pass, it takes encoded document representations (batch_doc_encodes) and the number of sentences in each document (batch_doc_sent_nums). It packs the input sequences to handle variable-length sequences efficiently, passes them through an LSTM layer, and unpacks the outputs. The final hidden state from the last timestep of the LSTM is used as input for a feedforward neural network (FNN). This FNN consists of customizable hidden layers and activations. It produces a single scalar value as the semantic score for each document, which represents the model's assessment of the semantic content in the input text. The choice of architecture and hyperparameters makes this module suitable for various tasks involving semantic scoring and the analysis of textual data.

```python
class SemanticScore(nn.Module):
    def __init__(self):
        super(SemanticScore, self).__init__()

        self.bert_emb_dim = 768
        self.dropout_prob = 0.5
        self.lstm_hidden_size = 1024
        self.lstm_layers_num = 1
        self.fnn_hidden_size = []
        self.bidirectional = False

        self.lstm = nn.LSTM(self.bert_emb_dim,
                            self.lstm_hidden_size,
                            self.lstm_layers_num,
                            bidirectional=self.bidirectional,
                            batch_first=True)
        self.dropout = nn.Dropout(self.dropout_prob)

        in_features = self.lstm_hidden_size * 2 if self.bidirectional else self.lstm_hidden_size
        layers = []
        for hs in self.fnn_hidden_size:
            layers.append(nn.Linear(in_features, hs))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(self.dropout_prob))
            in_features = hs

        layers.append(nn.Linear(in_features, 400))
        layers.append(nn.ReLU())
        layers.append(nn.Dropout(self.dropout_prob))
        layers.append(nn.Linear(400, 1))
        layers.append(nn.Sigmoid())
        self.fnn = nn.Sequential(*layers)

    def forward(self, batch_doc_encodes, batch_doc_sent_nums):
        packed_input = pack_padded_sequence(batch_doc_encodes, batch_doc_sent_nums, batch_first=True, enforce_sorte
        packed_output, _ = self.lstm(packed_input)
        output, _ = pad_packed_sequence(packed_output, batch_first=True)
        logits = self.fnn(output[:, -1, :]) # Using the output of the last timestep
        return logits.squeeze(-1)
```

```
: custom_df.head()
```

:

	essay_id	essay_set	essay	normalized_score	prompt
0	1	1	dear local newspaper i think effects computers...	6.0	More and more people use computers, but not ev...
1	2	1	dear caps1 caps2 i believe that using computer...	7.0	More and more people use computers, but not ev...
2	3	1	dear caps1 caps2 caps3 more and more people us...	5.0	More and more people use computers, but not ev...
3	4	1	dear local newspaper caps1 i have found that m...	8.0	More and more people use computers, but not ev...
4	5	1	dear location1 i know having computers has a p...	6.0	More and more people use computers, but not ev...

Coherence score: It is exploited to detect the essays composed of permuted paragraphs. For the first kind of adversarial samples, the essays containing permuted well-written paragraphs, we also utilize the coherence model to detect. Obviously, coherence scores for well-organized essays must be higher than the permuted essays.Our coherence model is LSTM/BI-LSTM based.
The CoherenceScore class is designed to calculate coherence scores for input text data, which typically assess the logical or thematic consistency of the text. It is initialized with hyperparameters like the dimension of BERT embeddings, LSTM settings, and dropout probabilities. In the forward pass, it takes encoded document representations (batch_doc_encodes) and the number of elements or sentences in each document (batch_doc_sent_nums). The input sequences are packed with pack_padded_sequence to handle variable-length data efficiently. The packed input is processed through an LSTM layer, and the resulting packed output is then unpacked with pad_packed_sequence. The output from the last timestep is passed through a feedforward neural network (FNN), although in this code, no hidden layers are defined in the FNN, resulting in a single linear layer for score prediction. The output is a single scalar value representing the coherence score, which reflects the model's assessment of the logical or thematic consistency in the input text, making it suitable for tasks involving coherence evaluation and analysis of textual data.

```python
class CoherenceScore(nn.Module):
    def __init__(self):
        super(CoherenceScore, self).__init__()

        self.bert_emb_dim = 768
        self.dropout_prob = 0.5
        self.lstm_hidden_size = 1024
        self.lstm_layers_num = 2
        self.fnn_hidden_size = []
        self.bidirectional = False

        self.lstm = nn.LSTM(self.bert_emb_dim,
                            self.lstm_hidden_size,
                            self.lstm_layers_num,
                            bidirectional=self.bidirectional,
                            batch_first=True)
        self.dropout = nn.Dropout(self.dropout_prob)

        in_features = self.lstm_hidden_size * 2 if self.bidirectional else self.lstm_hidden_size
        layers = []
        # for hs in self.fnn_hidden_size:
        #     layers.append(nn.Linear(in_features, hs))
        #     layers.append(nn.ReLU())
        #     layers.append(nn.Dropout(self.dropout_prob))
        #     in_features = hs

        layers.append(nn.Linear(in_features, 400))
        layers.append(nn.ReLU())
        layers.append(nn.Dropout(self.dropout_prob))
        layers.append(nn.Linear(400, 1))
        layers.append(nn.Sigmoid())
        self.fnn = nn.Sequential(*layers)

    def forward(self, batch_doc_encodes, batch_doc_sent_nums):
        packed_input = pack_padded_sequence(batch_doc_encodes, batch_doc_sent_nums, batch_first=True, enforce_sorte
        # print(packed_input.data.shape,"packed_input")
        packed_output, _ = self.lstm(packed_input)
        # print(packed_output.data.shape,"packed_out")
        output, _ = pad_packed_sequence(packed_output, batch_first=True)
        logits = self.fnn(output[:, -1, :]) # Using the output of the last timestep
        return logits.squeeze(-1)
```

	essay_id	essay_set	essay	normalized_score	prompt
23957	111976	8	laughter is good for every one.caps1 helps the...	0.0	We all understand the benefits of laughter. Fo...
23958	111977	8	due you think that being part of sameting funi...	0.0	We all understand the benefits of laughter. Fo...
23959	111978	8	i was raised with two brother that mean the wo...	0.0	We all understand the benefits of laughter. Fo...
23960	111979	8	its ture its good to laugh makes everyone a be...	0.0	We all understand the benefits of laughter. Fo...
23961	111980	8	i dont like computers	0.0	We all understand the benefits of laughter. Fo...

Prompt relevant score: The connections between prompts and essays are evaluated based on prompt-relevant scores, which are defined to detect the prompt-irrelevant samples. The prompt-relevant score is calculated based on LSTM/BI-LSTM again, and the details are illustrated in the above figure.
The PromptScore class is designed to compute prompt scores for text data related to specific prompts or questions. It initializes with hyperparameters, including BERT embedding dimensions, LSTM settings, and dropout probabilities. In the forward pass, it takes encoded input data (batch_doc_encodes) and the number of elements in each input (batch_doc_sent_nums). The code efficiently handles variable-length sequences by packing the input with pack_padded_sequence, passes them through an LSTM layer, and unpacks the outputs with pad_packed_sequence. The final hidden state from the last timestep of the LSTM is used as input for a feedforward neural network (FNN). This FNN consists of customizable hidden layers and activations and produces a single scalar value as the prompt score. This score reflects the model's assessment of the quality or relevance of responses to specific prompts or questions in the input text. The choice of architecture and hyperparameters makes this module suitable for various tasks involving prompt-based scoring or evaluation of textual data.

```python
class PromptScore(nn.Module):
    def __init__(self):
        super(PromptScore, self).__init__()
        self.bert_emb_dim = 768
        self.dropout_prob = 0.5
        self.lstm_hidden_size = 1024
        self.lstm_layers_num = 1
        self.fnn_hidden_size = []
        self.bidirectional = False

        self.lstm = nn.LSTM(self.bert_emb_dim,
                            self.lstm_hidden_size,
                            self.lstm_layers_num,
                            bidirectional=self.bidirectional,
                            batch_first=True)
        self.dropout = nn.Dropout(self.dropout_prob)

        in_features = self.lstm_hidden_size * 2 if self.bidirectional else self.lstm_hidden_size
        layers = []
        for hs in self.fnn_hidden_size:
            layers.append(nn.Linear(in_features, hs))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(self.dropout_prob))
            in_features = hs

        layers.append(nn.Linear(in_features, 400))
        layers.append(nn.ReLU())
        layers.append(nn.Dropout(self.dropout_prob))
        layers.append(nn.Linear(400, 1))
        layers.append(nn.Sigmoid())
        self.fnn = nn.Sequential(*layers)

    def forward(self, batch_doc_encodes, batch_doc_sent_nums):
        packed_input = pack_padded_sequence(batch_doc_encodes, batch_doc_sent_nums, batch_first=True, enforce_sorte
        packed_output, _ = self.lstm(packed_input)
        output, _ = pad_packed_sequence(packed_output, batch_first=True)
        logits = self.fnn(output[:, -1, :]) # Using the output of the last timestep
        return logits.squeeze(-1)
```

```
: final_prompt_df.tail()
```

:

	essay_id	essay_set	essay	normalized_score	prompt
35903	20967	8	Read the last paragraph of the story.\n "Wh...	0.0	We all understand the benefits of laughter. Fo...
35904	21389	8	Censorship in the Libraries\n All of us can...	0.0	We all understand the benefits of laughter. Fo...
35905	21389	8	More and more people use computers, but not ev...	0.0	We all understand the benefits of laughter. Fo...
35906	21389	8	Based on the excerpt, describe the obstacles t...	0.0	We all understand the benefits of laughter. Fo...
35907	21619	8	Read the last paragraph of the story.\n "Wh...	0.0	We all understand the benefits of laughter. Fo...

Handcrafted features (WORD COUNT,CHARACTER COUNT,MEAN LENGTH ,VARIANCE SCORE,GRAMMATICAL SCORE)

WORD COUNT:

- Definition: The total number of words present in a given text.
- Purpose: Reflects the document's overall length and can be indicative of content richness or verbosity.

CHARACTER COUNT:

- Definition: The total number of characters, including spaces, in a given text.
- Purpose: Measures the length of the text on a character basis, which can be relevant in certain contexts, such as character limits in social media or other platforms.

MEAN LENGTH:

- Definition: The average length of words in a given text, calculated by dividing the total number of characters by the number of words.

- Purpose: Provides insights into the average size of words, which can be relevant in stylistic or linguistic analyses.

VARIANCE SCORE:

- Definition: A measure of the variability or dispersion of word lengths within a text.
- Purpose: Indicates how spread out the word lengths are, helping to identify patterns or irregularities in the distribution of word lengths.

GRAMMATICAL SCORE:

- Definition: A quantitative measure reflecting the grammatical correctness or sophistication of a text.
- Purpose: Offers an assessment of the grammatical quality of the text, which can be useful in tasks such as automated essay grading or evaluating language proficiency.

```python
def get_grammatical_score(text, tool=tool):
    size = len(text.split())
    num = len(tool.check(text))
    # print(size, num)
    return (size - num) / size

def get_word_count(text):
    return len(text.split())

def get_char_count(text):
    return len(text)

def get_mean_score(text):
    words = text.split()
    word_lengths = [len(word) for word in words]
    mean_word_length = sum(word_lengths) / len(word_lengths)
    return mean_word_length

def get_variance_score(text):
    words = text.split()
    word_lengths = [len(word) for word in words]
    mean_word_length = sum(word_lengths) / len(word_lengths)
    variance_word_length = sum((length - mean_word_length) ** 2 for length in word_lengths) / len(word_lengths)
    return variance_word_length

print(get_mean_score(text))
print(get_variance_score(text))
print(get_grammatical_score(text))
print(get_word_count(text) )
print(get_char_count(text))
```

```
4.43026706231454
5.924662539953674
0.8783382789317508
337
1829
```

NOVELITY:

Proposed Architecture (For semantic score)

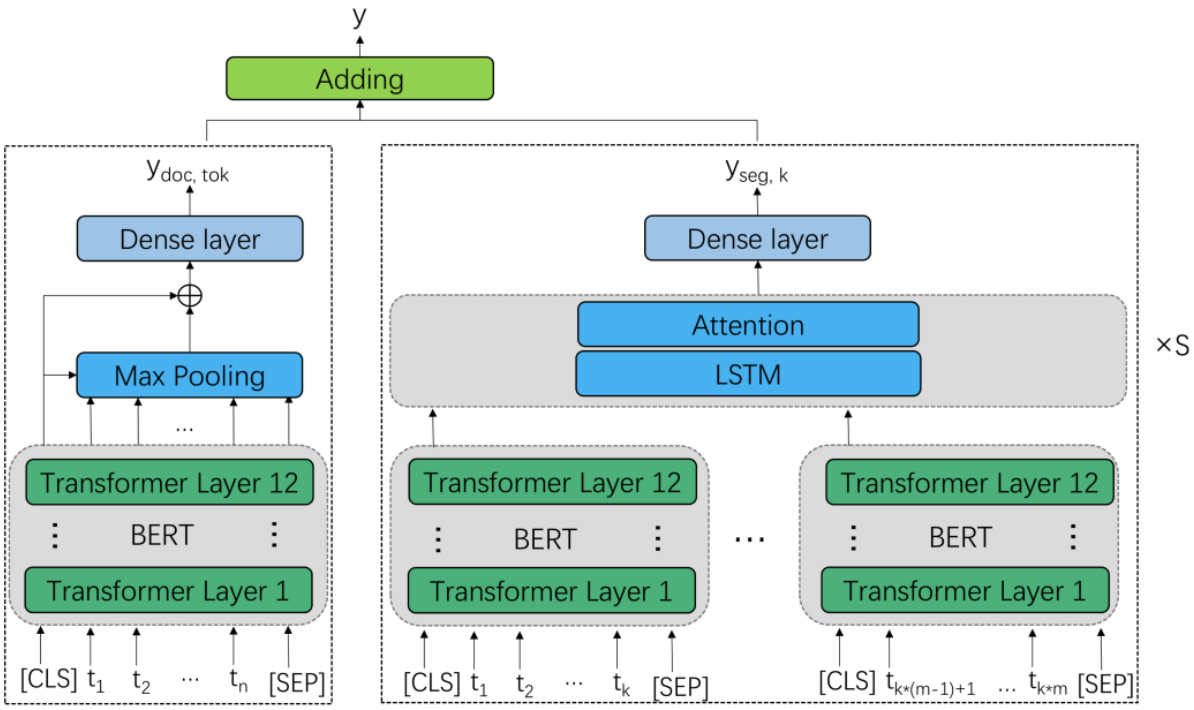


Figure 1: The proposed automated essay scoring architecture based on multi-scale essay representation. The left part illustrates the document-scale and token-scale essay representation and scoring module, and the right part illustrates S segment-scale essay representations and scoring modules.

1. **Document and token level representation :**

The provided model, named EssayBERTModel_, is a PyTorch implementation of a BERT-based neural network for regression tasks. The model's purpose is to take input essays, tokenize them using the BERT tokenizer, and then pass the tokenized input through a pre-trained BERT model (in

this case, 'bert-base-uncased'). The model utilizes the outputs of the BERT model, specifically the last hidden states and the pooler output, to create both token-scale and document-scale representations of the input essays. These representations are then concatenated and passed through a regression layer consisting of linear layers and ReLU activation functions. The final output scores from the regression layer serve as the model's predictions for the regression task, with the architecture designed to capture contextual information from the essays and produce meaningful regression results. The model incorporates frozen BERT parameters and a configurable regression output size,

```
class EssayBERTModel_(nn.Module):
    def __init__(self, bert_model_name='bert-base-uncased', regression_output_size=1):
        super(EssayBERTModel_, self).__init__()

        # BERT model and tokenizer
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.tokenizer = BertTokenizer.from_pretrained(bert_model_name)

        for param in self.bert.parameters():
            param.requires_grad = False

        # Regression layer with dropout
        self.regression_layer = nn.Sequential(
            nn.Linear(2 * self.bert.config.hidden_size, 512),
            nn.ReLU(),
            # nn.Dropout(0.1),
            nn.Linear(512, regression_output_size)
        )

    def forward(self,essays):
        # Tokenize input essay

        tokenized_input = tokenizer(essays, return_tensors='pt', max_length=512, truncation=True, padding='max_length')

        input_ids = tokenized_input['input_ids']
        attention_mask = tokenized_input['attention_mask']

        batch_size, max_tokens = input_ids.size()

        # Add an extra dimension for num_segments
        input_ids = input_ids.unsqueeze(1)
        attention_mask = attention_mask.unsqueeze(1)

        # Reshape input_ids and attention_mask for BERT processing
        input_ids_flat = input_ids.view(batch_size * 1, max_tokens)
        attention_mask_flat = attention_mask.view(batch_size * 1, max_tokens)

        # Step 1: BERT Processing
        outputs = self.bert(input_ids_flat, attention_mask=attention_mask_flat)

        # Max pooling over the sequence outputs for token-scale representation
        token_representation, _ = torch.max(outputs.last_hidden_state, dim=1)
```

2. **Segment scale representation:**

The SegmentScaleEssayModelo is a PyTorch neural network tailored for essay processing, combining BERT, LSTM, and attention mechanisms for segment-scale representations. It takes tokenized essays and attention masks as input, employing a pre-trained BERT model for initial processing and an LSTM layer to capture sequential dependencies within each essay segment. Attention pooling is applied to the LSTM outputs, generating segment-scale representations for each essay segment. The model utilizes separate dense regression layers for each specified segment scale, mapping the segment-scale representations to regression scores. The final output is calculated by averaging these scores across segment scales, providing a dynamic and context-aware approach for predicting continuous output scores in regression tasks. The architecture is adaptable to varying segment scales, demonstrating flexibility in handling essays of different lengths and structures.

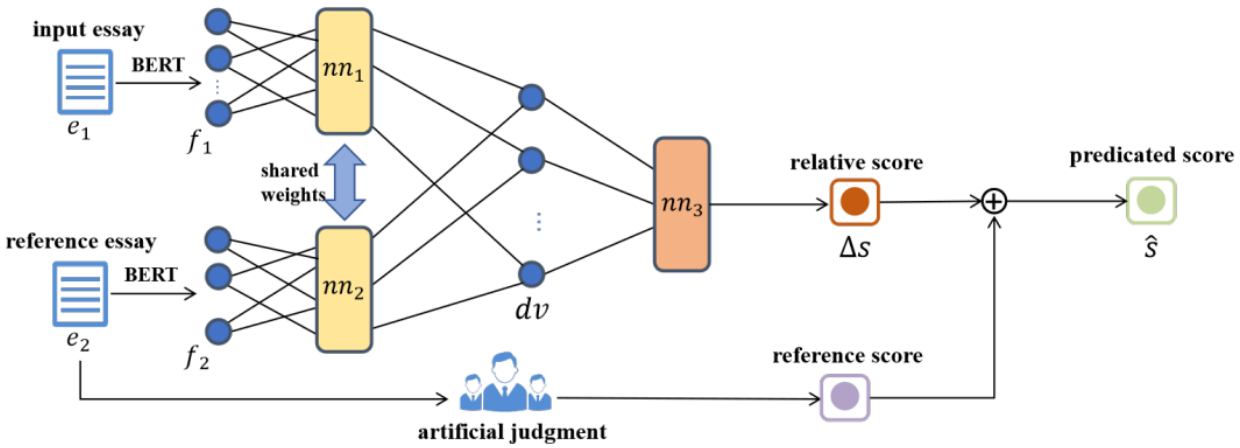
```python
class SegmentScaleEssayModelo(nn.Module):
    def __init__(self, bert_model, lstm_hidden_size, segment_scales):
        super(SegmentScaleEssayModelo, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model)
        self.lstm = nn.LSTM(input_size=768, hidden_size=lstm_hidden_size, batch_first=True, dropout=0.1)
        self.attention_pooling = nn.Linear(lstm_hidden_size, 1)
        self.tanh = nn.Tanh()
        self.softmax = nn.Softmax(dim=1)
        self.segment_scales = segment_scales
        self.lstm_hidden=lstm_hidden_size

        for param in self.bert.parameters():
             param.requires_grad = False
        # Create dense regression layers for each segment-scale with dropout
        self.regression_layers = nn.ModuleList([nn.Sequential(
            nn.Linear(lstm_hidden_size, 512),  # Adjust the size as needed
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(512, 1)
        ) for _ in segment_scales])

    def forward(self, input_ids, attention_mask):

        batch_size, num_segments, max_tokens = input_ids.size()

        # Reshape input_ids and attention_mask for BERT processing
        input_ids_flat = input_ids.view(batch_size * num_segments, max_tokens)
        attention_mask_flat = attention_mask.view(batch_size * num_segments, max_tokens)

        # Step 1: BERT Processing
        outputs = self.bert(input_ids_flat, attention_mask=attention_mask_flat)
        sequence_outputs = outputs.last_hidden_state

        # Reshape sequence_outputs back to 3D
        sequence_outputs = sequence_outputs.view(batch_size, num_segments, max_tokens, -1)

        # Initialize a list to store segment outputs
        segment_outputs = []

        for segment_index in range(num_segments):
            # Select the current segment from the 3D tensor
            current_segment = sequence_outputs[:, segment_index, :, :]
```

3. Combined model

The CombinedEssayModel_ is a PyTorch neural network that integrates two distinct models, the SegmentScaleEssayModelo and the EssayBERTModel_, to provide a combined representation for essay processing. The model takes as input the raw input_text and the maximum number of tokens (max_tokens). It first tokenizes the input text using the BERT tokenizer and then utilizes both the segment-scale model (self.segment_scale_model) and the document-scale and token-scale model (self.essay_bert_model) to obtain their respective representations. The segment-scale representation is obtained through the segment_scale_representation method of the SegmentScaleEssayModelo, while the document-scale and token-scale representation are obtained through the EssayBERTModel_. The final combined representation is calculated by adding the scores from both models. This architecture allows the model to capture both segment-scale and overall essay characteristics, making it suitable for tasks requiring a comprehensive understanding of essays, such as regression problems where continuous output scores are predicted.

```python
class CombinedEssayModel_(nn.Module):
    def __init__(self, bert_model, lstm_hidden_size, segment_scales, regression_output_size=1):
        super(CombinedEssayModel_, self).__init__()

        self.tokenizer=BertTokenizer.from_pretrained('bert-base-uncased')

        # Segment-scale model
        self.segment_scale_model = SegmentScaleEssayModelo(bert_model, lstm_hidden_size, segment_scales)

        # Document-scale and Token-scale model
        self.essay_bert_model = EssayBERTModel_(bert_model, regression_output_size)

    def forward(self, input_text, max_tokens):

        tokenized_input = tokenizer(input_text, return_tensors='pt', max_length=max_tokens, truncation=True, padding='max_length')

        input_ids = tokenized_input['input_ids']
        attention_mask = tokenized_input['attention_mask']
        # Get segment-scale representation
        segment_scale_representation_score = self.segment_scale_model.segment_scale_representation(input_ids, attention_mask)

        # print(segment_scale_representation_score)
        # Get document-scale and token-scale representation
        essay_representation_score = self.essay_bert_model(input_text)

        # print(essay_representation_score)
        score=essay_representation_score+segment_scale_representation_score

        return score
```

4. Pairwise Contrasitive Regression model:



The npcr_model is a PyTorch neural network designed for Pairwise Contrasitive Regression tasks, specifically computing the Normalized Pointwise Cross-Region (NPCR) score. The model utilizes a pre-trained BERT model (bert_model) for embedding pairs of input sequences (x0 and x1). The embeddings are processed independently through linear layers (nn1), followed by dropout for regularization. The pointwise difference between the processed embeddings is then computed. The resulting difference is passed through a linear layer (output) with a sigmoid activation function, indicating a binary classification task where the model predicts the similarity or dissimilarity between the input pairs. The model is initialized using specific weight initialization

strategies, and the architecture is tailored for contrastive learning, aiming to learn representations that effectively capture the pairwise relationships between input sequences for tasks like Pairwise Contrasitive Regression.

```
class npcr_model(nn.Module):
    def __init__(self, maxSq=512):
        super(npcr_model, self).__init__()

        self.embedding = bert_model
        self.dropout = nn.Dropout(0.5)

        self.nn1 = nn.Linear(768, 768)
        self.output = nn.Linear(768, 1, bias=False)

        self.init_weights()

    def init_weights(self):
        """
        Here we reproduce Keras default initialization weights for consistency with Keras version
        """
        ih = (param.data for name, param in self.named_parameters() if 'weight_ih' in name)
        hh = (param.data for name, param in self.named_parameters() if 'weight_hh' in name)
        b = (param.data for name, param in self.named_parameters() if 'bias_ih' in name or 'bias_hh' in name)
        # nn.init.uniform(self.embed.weight.data, a=-0.5, b=0.5)
        for t in ih:
            nn.init.xavier_uniform_(t)
        for t in hh:
            nn.init.orthogonal_(t)
        for t in b:
            nn.init.constant_(t, 0)

    def forward(self, x0, x1):
        x0_embed = self.embedding(x0)[1]
        x1_embed = self.embedding(x1)[1]

        # the linear layer nn1 can be replaced by MLP(the above or overwite by yourself)
        x0_nn1 = self.nn1(x0_embed)
        x1_nn1 = self.nn1(x1_embed)

        x0_nn1_d = self.dropout(x0_nn1)
        x1_nn1_d = self.dropout(x1_nn1)

        diff_x = (x0_nn1_d - x1_nn1_d)
        y = self.output(diff_x)

        y = torch.sigmoid(y)
```

5. **Semantic score model:**

The BERTNLIModel is a PyTorch neural network designed for predicting semantic scores, particularly in the context of Natural Language Inference (NLI) tasks. The model integrates a pre-trained BERT model (bert_model) to leverage its contextualized embeddings, capturing nuanced semantic information from input sequences. The BERT model takes input sequences (sequence), attention masks (attn_mask), and token type IDs (token_type) and outputs a pooled representation. This representation is then passed through a linear output layer (out) to produce a continuous semantic score, representing the degree of semantic similarity or relatedness between pairs of input sequences. The model is structured for tasks where understanding the semantic relationship between sentences is crucial, making it suitable for applications like semantic textual similarity or related semantic scoring.

```
class BERTNLIModel(nn.Module):
  def __init__(self, bert_model, hidden_dim, output_dim,):
    super().__init__()
    self.bert = bert_model
    embedding_dim = bert_model.config.to_dict()['hidden_size']
    self.out = nn.Linear(embedding_dim, output_dim)

  def forward(self, sequence, attn_mask, token_type):
    embedded = self.bert(input_ids = sequence, attention_mask =
                         attn_mask, token_type_ids = token_type)[1]
    output = self.out(embedded)
    return output
```

SECOND STAGE: In the second stage, we give weightage to these three scores with some handcrafted features, and the results are fed into the eXtreme Gradient Boosting model (XGboost) for further training.

We have use 5 handcrafted features in our project i.e, grammatical score,word count,character count,mean score and variance score.

```
text = "Your the best but their are allso  good abvfgc!"

def get_grammatical_score(text, tool=tool):
    size = len(text.split())
    num = len(tool.check(text))
    # print(size, num)
    return (size - num) / size

def get_word_count(text):
    return len(text.split())

def get_char_count(text):
    return len(text)

def get_mean_score(text):
    words = text.split()
    word_lengths = [len(word) for word in words]
    mean_word_length = sum(word_lengths) / len(word_lengths)
    return mean_word_length

def get_variance_score(text):
    words = text.split()
    word_lengths = [len(word) for word in words]
    mean_word_length = sum(word_lengths) / len(word_lengths)
    variance_word_length = sum((length - mean_word_length) ** 2 for length in word_lengths) / len(word_lengths)
    return variance_word_length

print(get_mean_score(text))
print(get_variance_score(text))
print(get_grammatical_score(text))
print(get_word_count(text) )
print(get_char_count(text))
```

```
4.222222222222222
1.506172839506173
0.4444444444444444
9
47
```

eXtreme Gradient Boosting model (XGboost):

```python
import xgboost as xgb

dtrain = xgb.DMatrix(X_train, label=y_train)
dvalid = xgb.DMatrix(X_valid, label=y_valid)

param = {
    'max_depth': 3,
    'eta': 0.01,
    'objective': 'reg:squarederror'
}
num_round = 100

bst = xgb.train(param, dtrain, num_round, [(dtrain, 'train'), (dvalid, 'valid')])
```

```
[0]     train-rmse:2.52929      valid-rmse:2.29797
[1]     train-rmse:2.51520      valid-rmse:2.29071
[2]     train-rmse:2.50125      valid-rmse:2.28362
[3]     train-rmse:2.48743      valid-rmse:2.27669
[4]     train-rmse:2.47373      valid-rmse:2.26993
[5]     train-rmse:2.46017      valid-rmse:2.26333
[6]     train-rmse:2.44673      valid-rmse:2.25690
[7]     train-rmse:2.43342      valid-rmse:2.25062
[8]     train-rmse:2.42024      valid-rmse:2.24450
[9]     train-rmse:2.40718      valid-rmse:2.23854
[10]    train-rmse:2.39424      valid-rmse:2.23273
[11]    train-rmse:2.38143      valid-rmse:2.22707
[12]    train-rmse:2.36873      valid-rmse:2.22157
[13]    train-rmse:2.35562      valid-rmse:2.21921
[14]    train-rmse:2.34265      valid-rmse:2.21694
[15]    train-rmse:2.32981      valid-rmse:2.21476
[16]    train-rmse:2.31711      valid-rmse:2.21266
[17]    train-rmse:2.30454      valid-rmse:2.21065
[18]    train-rmse:2.29210      valid-rmse:2.20872
[19]    train-rmse:2.27980      valid-rmse:2.20687
[20]    train-rmse:2.26762      valid-rmse:2.20501
[21]    train-rmse:2.25556      valid-rmse:2.20331
[22]    train-rmse:2.24363      valid-rmse:2.20160
[23]    train-rmse:2.23182      valid-rmse:2.20005
[24]    train-rmse:2.21830      valid-rmse:2.20046
[25]    train-rmse:2.20670      valid-rmse:2.19894
[26]    train-rmse:2.19340      valid-rmse:2.19941
```

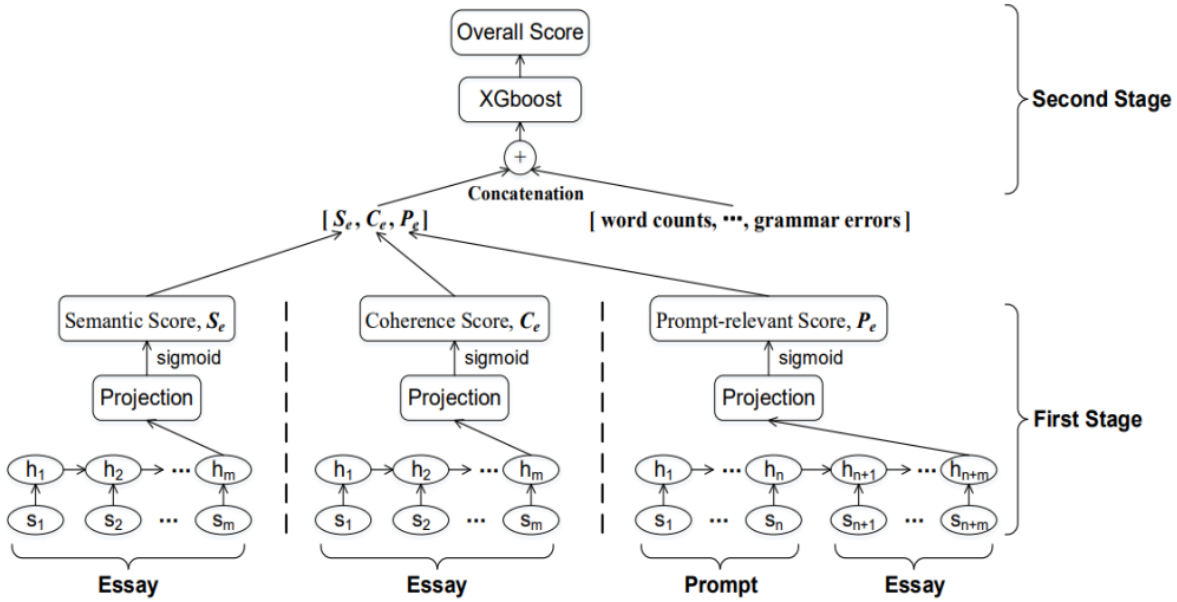
This code employs the XGBoost library to train a gradient boosting regression model. It begins by preparing training and validation datasets, with X_train and y_train representing features and labels for training, and X_valid and y_valid for validation. The code creates XGBoost DMatrix objects for these datasets. Next, model hyperparameters, such as the maximum tree depth and learning rate, are specified in the param dictionary, and the objective is set for regression with squared error loss. The model is trained with 100 boosting rounds using the xgb.train function, which takes the parameters, training data, and validation data for monitoring. Upon training completion, the trained XGBoost model is stored in the bst variable.

```python
def get_final_score(essays, prompt):
    main_data = pd.DataFrame()

    means, variances, grammaticals, word_counts, char_counts = get_mean_score(essays[0]), get_variance_score(essays[0]), get_grammatical_score(essays[0]),get_word_count(essays

    val = []
    for i in range(1):
        val.append(prompt[i]+"."+essays[i])

    val, lengths_batch = preprocess_essay(val)
    val = val.to(device)
    essays, lengths_batch = preprocess_essay(essays)
    essays = essays.to(device)
    out_semantic = model_semantic(essays, lengths_batch)
    out_coher = model_coher(essays, lengths_batch)

    out_prompt = model_prompt(val, lengths_batch)
    out_semantic = out_semantic.cpu().detach().numpy()
    out_coher = out_coher.cpu().detach().numpy()
    out_prompt = out_prompt.cpu().detach().numpy()

    main_data = pd.DataFrame()
    temp_df = pd.DataFrame({
        'means': means,
        'variances': variances,
        'grammaticals': grammaticals,
        'word_counts': word_counts,
        'char_counts': char_counts,
        'out_semantic': out_semantic[0],
        'out_coher': out_coher[0],
        'out_prompt': out_prompt[0],
        # 'normalized_score': normalized_score
      }, index=[0])

    main_data = pd.concat([main_data, temp_df], ignore_index=True)
```

The get_final_score function calculates multiple scores for essays. It first initializes lists to store mean, variance, grammatical, word count, and character count scores. Then, for a single essay

(though designed for multiple), it calculates and appends these scores. It combines each essay with a prompt and preprocesses the data for further analysis. Predictions for semantic, coherence, and prompt scores are made using specific models. The function retrieves these predictions, converts them to NumPy arrays, and compiles all the calculated and predicted scores into a list. This list is returned, providing a comprehensive assessment of essays, including linguistic and semantic attributes, and their relevance to a given prompt or question.

PROPOSED ARCHITECTURE DESIGN:



Other Novelities:

1) **Differentiate essay based on human written or ChatGPT:**

Creating features to differentiate between essays written by ChatGPT and those written by humans involves considering characteristics that may distinguish between the two. Here are 20 potential features:

Word Count:
- Compare the average word count per essay, as ChatGPT may generate responses with different lengths than typical human essays.

Sentence Structure Complexity:
- Analyze the syntactic complexity and variety of sentence structures, as humans might exhibit more diverse sentence structures.

Vocabulary Richness:
- Assess the diversity and richness of vocabulary in the essays, as humans may use a wider range of words and expressions.

Grammatical Errors:
- Count the number of grammatical errors to identify potential differences in language proficiency.

Use of Pronouns:
- Examine the frequency and distribution of pronouns, as ChatGPT might display different patterns in pronoun usage.

Paragraph Length:
- Compare the average length of paragraphs, as humans may organize ideas differently than ChatGPT.

Cohesiveness:
- Evaluate the cohesiveness of the text to identify how well ideas are connected within and between sentences.

Consistency of Tone:
- Assess the consistency of tone and style throughout the essays, as humans may maintain a more consistent voice.

Use of Specific References:
- Analyze the inclusion of specific references or details, as humans may draw on personal experiences or examples.

Contextual Understanding:
- Evaluate the ability to demonstrate a deep understanding of context and nuances in the essays.

Logical Flow:
- Assess the logical flow of ideas and arguments, as humans may structure their essays more coherently.

Use of Humor or Creativity:
- Identify instances of humor or creativity, as these elements may be expressed differently by ChatGPT and humans.

Awareness of Current Events:
- Check for references to recent events or real-world context, as humans may be more up-to-date with current affairs.

Emotional Expression:

- Analyze the expression of emotions, as humans may convey feelings and experiences in a more authentic manner.

Use of References:

- Evaluate the inclusion of citations or references, as humans may draw on external sources more deliberately.

Repetition Patterns:

- Examine repetition patterns in word choice or phrases, as ChatGPT may exhibit different patterns in generating content.

Handling Ambiguity:

- Assess how well ambiguity is handled in the essays, as humans may navigate ambiguity more effectively.

Introduction and Conclusion Quality:

- Evaluate the quality of introductions and conclusions, as humans may structure these sections differently.

Consistency in Argumentation:

- Check for consistency in argumentation and the development of ideas, as humans may present more cohesive arguments.

Engagement with the Prompt:

- Assess how well the essays engage with and directly address the given prompts, as humans may interpret prompts more contextually.

These features aim to capture a range of linguistic, stylistic, and contextual aspects that may differentiate between essays written by ChatGPT and those authored by humans. Depending on the specific characteristics of interest, additional or refined features may be considered.

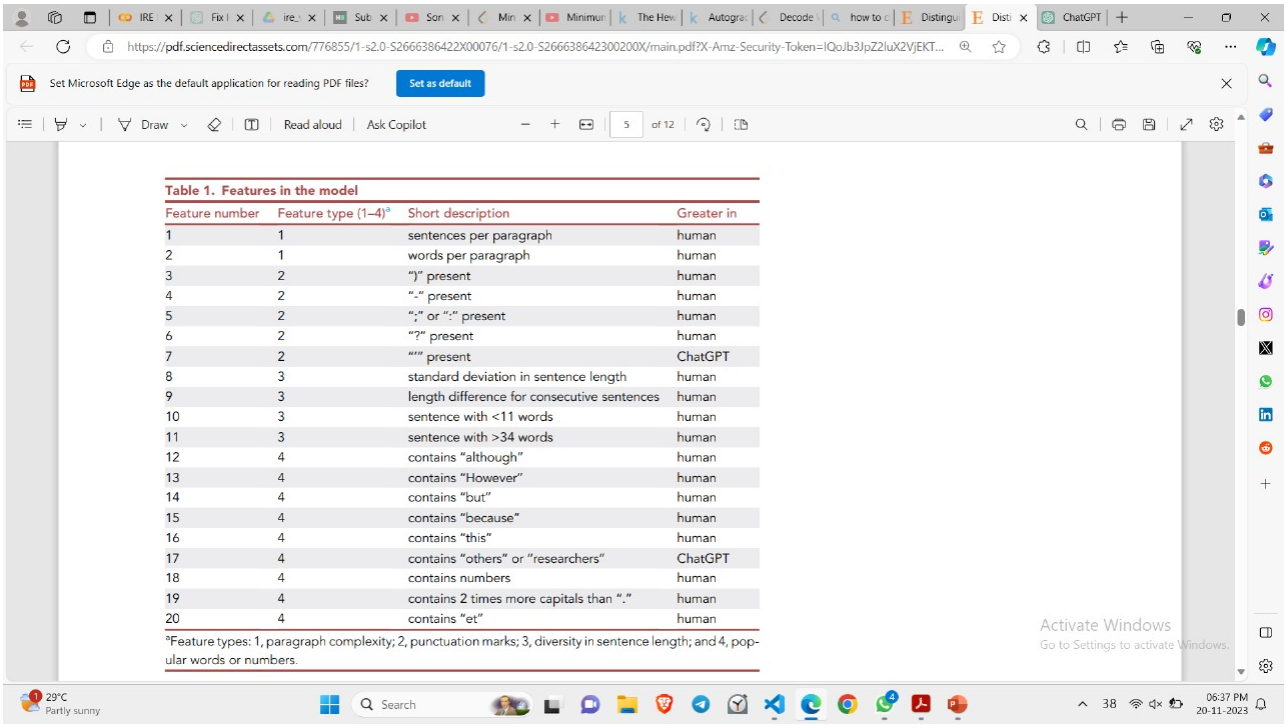


Table 1. Features in the model

Feature number	Feature type (1–4)[a]	Short description	Greater in
1	1	sentences per paragraph	human
2	1	words per paragraph	human
3	2	")" present	human
4	2	"-" present	human
5	2	";" or ":" present	human
6	2	"?" present	human
7	2	"'" present	ChatGPT
8	3	standard deviation in sentence length	human
9	3	length difference for consecutive sentences	human
10	3	sentence with <11 words	human
11	3	sentence with >34 words	human
12	4	contains "although"	human
13	4	contains "However"	human
14	4	contains "but"	human
15	4	contains "because"	human
16	4	contains "this"	human
17	4	contains "others" or "researchers"	ChatGPT
18	4	contains numbers	human
19	4	contains 2 times more capitals than "."	human
20	4	contains "et"	human

[a]Feature types: 1, paragraph complexity; 2, punctuation marks; 3, diversity in sentence length; and 4, popular words or numbers.

Evaluation Metrics:

1. Kendall's tau
   Definition: Kendall's tau is a measure of the correlation between two sets of rankings.
   Range: The value of Kendall's tau ranges from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.
   Function Explanation:
   a. The function uses the kendalltau function from the scipy.stats module to calculate Kendall's tau.
   b. The [0] indexing is used to extract the actual correlation coefficient from the result.
   c. The function returns the computed Kendall's tau, but if the result is NaN (Not a Number), it returns 0.0.

2. Spearman's Rank Correlation (spearman):
   Definition : Spearman's rank correlation is another measure of the monotonic relationship between two variables. It assesses how well the relationship between the variables can be described using a monotonic function.

Range: Similar to Kendall's tau, the value of Spearman's rank correlation ranges from -1 to 1.
Function Explanation:

a. The function uses the spearmanr function from the scipy.stats module to compute Spearman's rank correlation.
b. It extracts the actual correlation coefficient using [0] indexing.
c. If the result is NaN, it returns 0.0.

3. Pearson Correlation (pearson):
Definition: Pearson correlation measures the linear correlation between two variables. It quantifies how well the relationship between the variables can be described by a straight line.
Range: The value of Pearson correlation ranges from -1 to 1, where 1 indicates a perfect positive linear correlation, -1 indicates a perfect negative linear correlation, and 0 indicates no linear correlation.
Function Explanation:

a. The function uses the pearsonr function from the scipy.stats module to calculate Pearson correlation.
b. It extracts the actual correlation coefficient using [0] indexing.
c. If the result is NaN, it returns 0.0.

```python
from scipy.stats import kendalltau,spearmanr,pearsonr

def kendall_tau(y_true, y_pred):
    return kendalltau(y_true, y_pred)[0]

def spearman(y_true, y_pred):
    return spearmanr(y_true, y_pred)[0]

def pearson(y_true, y_pred):
    return pearsonr(y_true, y_pred)[0]


kendall_ta = kendall_tau(y_true, y_pred)
pearson_corr = pearson(y_true, y_pred)
spearma_corr = spearman(y_true, y_pred)


results_df = pd.DataFrame({
    'Kendall Tau': [kendall_ta],
    'Pearson Correlation': [pearson_corr],
    'Spearman Correlation': [spearma_corr],
})

print(results_df)
```

✓ 0.0s

	Kendall Tau	Pearson Correlation	Spearman Correlation
0	0.633349	0.728448	0.762087

RESULT:

We have tested our model on an essay, given a prompt,l and predicted its score.

PREDICTED SCORE VS REAL SCORE:

```python
y_true = np.array(y_valid)
y_pred = np.array(y_pred)
for i in range(len(y_pred)):
    print(f"Predicted Score : {y_pred[i]} -> Real Score :  {y_true[i]}")
```

✓ 0.0s

```
Predicted Score : 0.4690168797969818 -> Real Score :  0.5
Predicted Score : 0.6528865694999695 -> Real Score :  0.75
Predicted Score : 0.6940044164657593 -> Real Score :  0.6666666666666666
Predicted Score : 0.638420581817627 -> Real Score :  0.7
Predicted Score : 0.685191810131073 -> Real Score :  0.75
Predicted Score : 0.6284845471382141 -> Real Score :  0.6666666666666666
Predicted Score : 0.7264323830604553 -> Real Score :  0.75
Predicted Score : 0.611031174659729 -> Real Score :  0.6666666666666666
Predicted Score : 0.6649928092956543 -> Real Score :  0.6666666666666666
Predicted Score : 0.4747818112373352 -> Real Score :  0.5
Predicted Score : 0.6425468325614929 -> Real Score :  0.7
Predicted Score : 0.6747459769248962 -> Real Score :  0.75
Predicted Score : 0.6940044164657593 -> Real Score :  0.75
Predicted Score : 0.7264323830604553 -> Real Score :  0.75
Predicted Score : 0.6335738897323608 -> Real Score :  0.6666666666666666
Predicted Score : 0.657714307308197 -> Real Score :  0.6666666666666666
Predicted Score : 0.674974799156189 -> Real Score :  0.6
Predicted Score : 0.3850401043891907 -> Real Score :  0.3333333333333333
Predicted Score : 0.685191810131073 -> Real Score :  0.75
Predicted Score : 0.6131255030632019 -> Real Score :  0.6
Predicted Score : 0.5867252945899963 -> Real Score :  0.6666666666666666
Predicted Score : 0.6009930968284607 -> Real Score :  0.6
Predicted Score : 0.6649928092956543 -> Real Score :  0.75
Predicted Score : 0.6369709372520447 -> Real Score :  0.6
Predicted Score : 0.37317758798599243 -> Real Score :  0.3333333333333333
```

GIVEN A ESSAY AND PROMPT WE ARE PREDICTING ITS SCORE

```
idx = 5
ess = [custom_df['essay'].iloc[idx]]
prp = [custom_df['prompt'].iloc[idx]]

print(ess)
print(prp)
output = get_final_score(ess, prp)
# print("OUTPUT : ", output.shape)
dpredict = xgb.DMatrix(output)
y_pred = bst.predict(dpredict)
print(np.round(y_pred*10))
```

✓ 2.2s

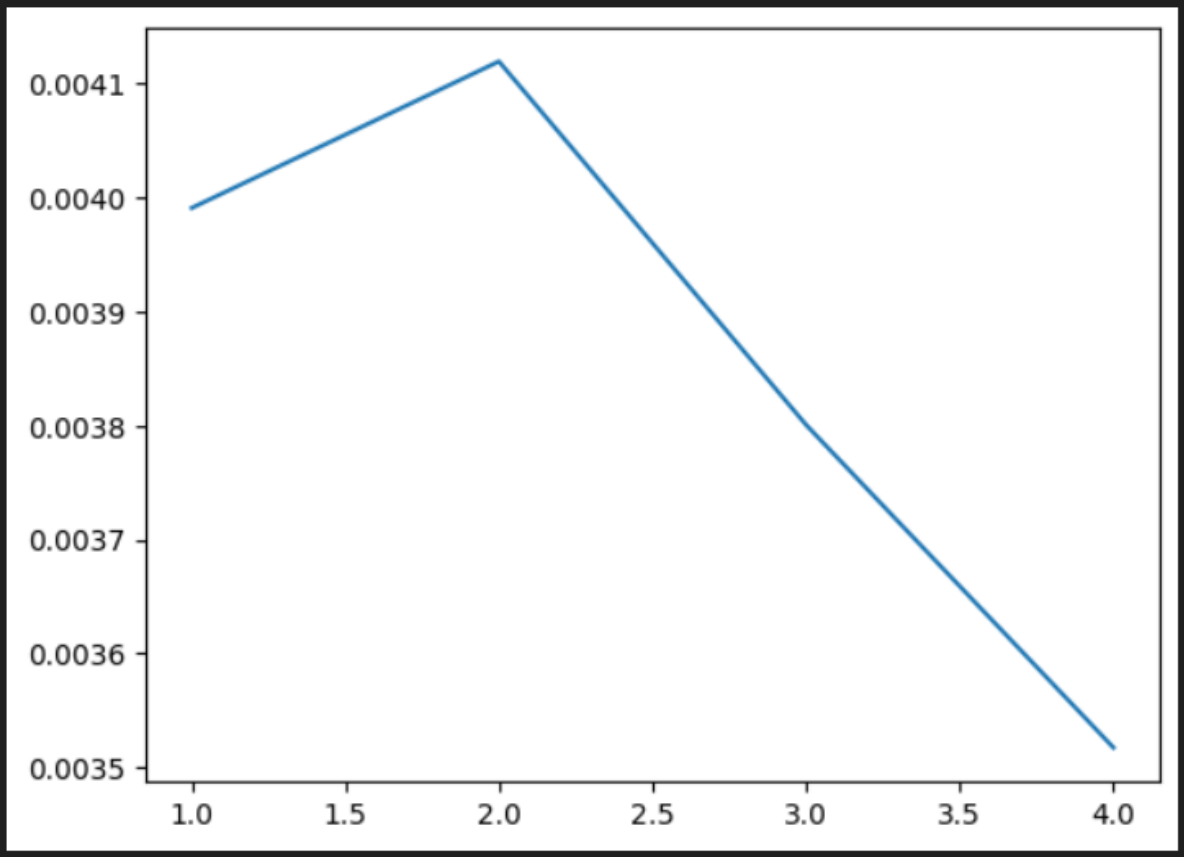
```
['dear location1 i think that computers have a negative affect on us how many people have acess to a camputer daily in america.. num1 and how many people go on at lea
['More and more people use computers, but not everyone agrees that this benefits society.\n    Those who support advances in technology believe that computers have a
[6.]
```

PROMPT MODEL LOSS

```
epochs=[1,2,3,4]
plt.plot(epochs,train_loss_[:4])
```

✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x1f72ecbc670>]
```

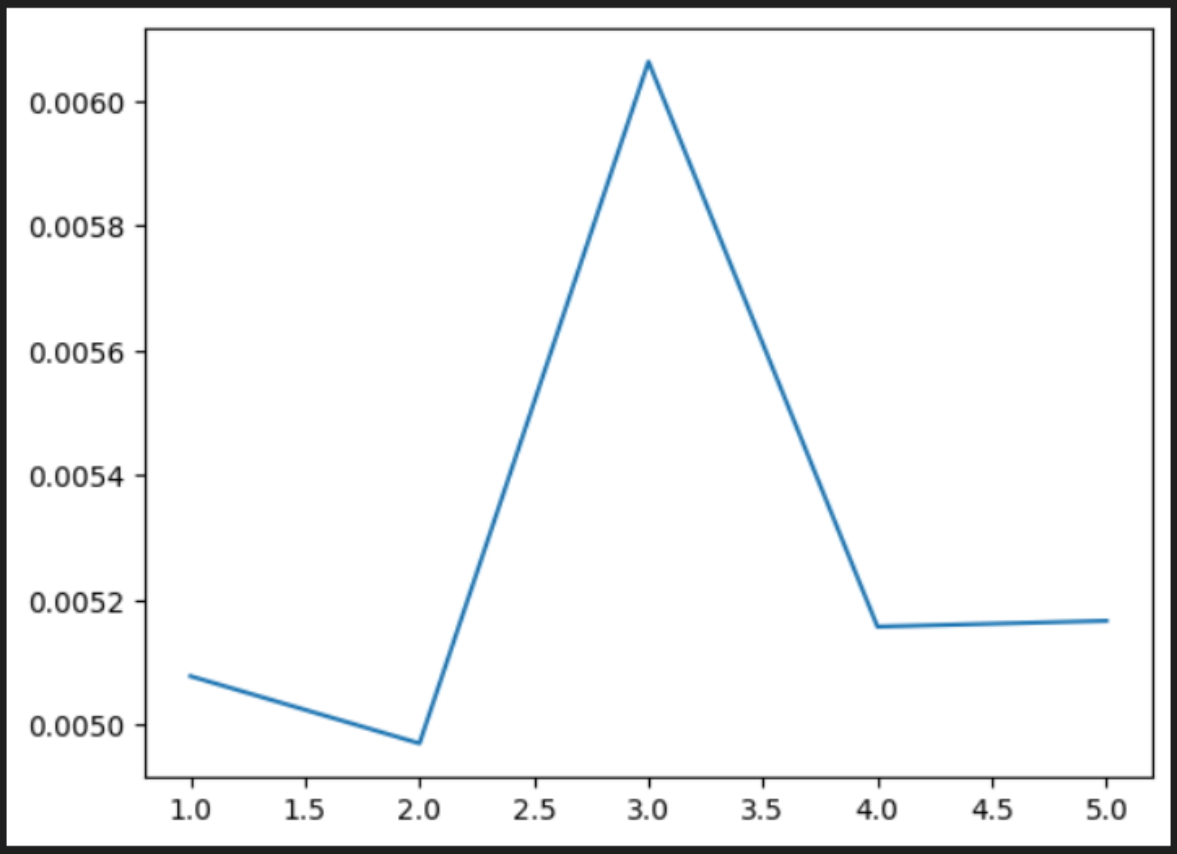


COHERENCE MODEL LOSS

```
epochs=[1,2,3,4,5]
plt.plot(epochs,train_loss_coher)
```

✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x1f72ed36520>]
```

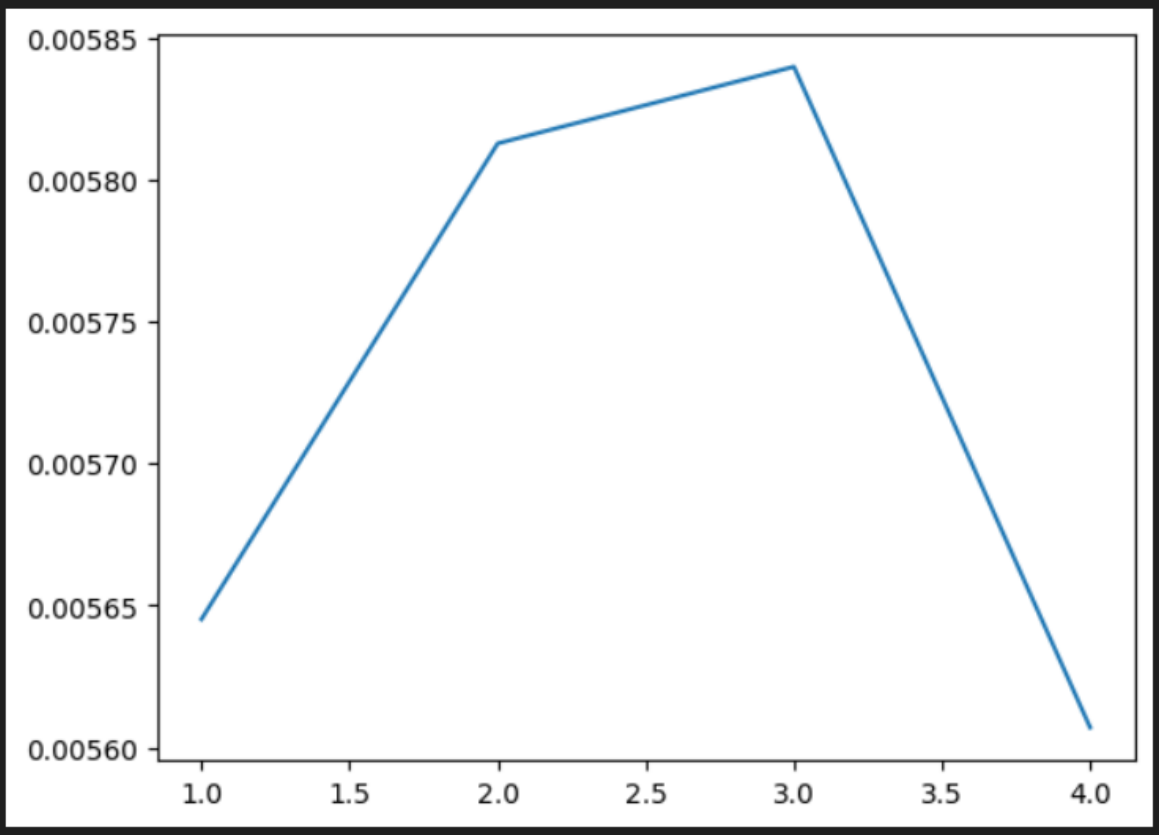


SEMANTIC MODEL LOSS

```
epochs=[1,2,3,4]
plt.plot(epochs,train_loss[:4])
```

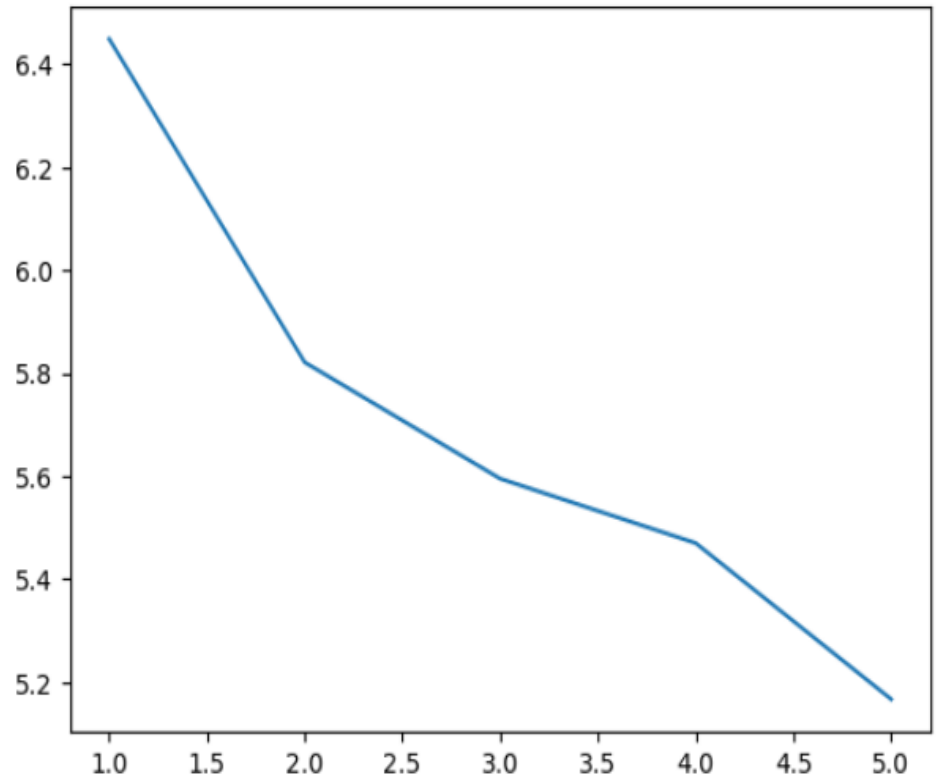
✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x1f72ee4ea30>]
```



COMBINED MODEL LOSS:

```
In [ ]: plt.plot([1,2,3,4,5],train_loss_combine)
Out[77]: [<matplotlib.lines.Line2D at 0x212595e4340>]
```



DATAFRAME GENERATED FOR XG-BOOST TRAINING OF NOVELITY

147]: (X_valid)

147]:

	means	variances	grammaticals	word_counts	char_counts	out_semantic	out_coher	out_prompt	combined_score	coherence_score_nli
213	4.954955	7.196169	0.891892	111	660	0.500796	0.503431	0.346328	0.684430	0.310375
331	5.120690	6.433710	0.887931	116	709	0.501566	0.483666	0.368868	0.784117	0.175243
501	4.313043	4.690410	0.939130	345	1832	0.504080	0.292549	0.346097	0.565251	0.143300
309	4.830409	6.374748	0.918129	171	996	0.503389	0.489085	0.369265	0.872319	0.098921
88	3.328125	2.230876	0.838542	192	830	0.503660	0.233364	0.381834	0.595867	0.170512
...	...	...	...	...	...	...	...	...	...	...
705	4.257840	4.435261	0.846690	287	1508	0.502187	0.507885	0.002818	0.535295	0.203311
305	4.990566	6.518779	0.896226	106	634	0.500003	0.509265	0.352399	0.494369	0.062507
809	4.138462	3.780828	0.923077	65	333	0.504424	0.266739	0.377860	0.428078	0.015809
237	3.828169	3.308502	0.847887	355	1713	0.504365	0.512918	0.050550	0.520719	0.327485
754	5.130435	5.635161	0.847826	46	281	0.502735	0.493073	0.000107	0.241110	0.205307

NOVELITIES:

FIRST COMBINED MODEL:

We used coherence score_nli and ranks combined with few models from framework

COMBINED MODEL 4

```
X_combined_4 = final_dataframe[[ 'out_prompt', 'combined_score', 'coherence_score_nli', 'ranks']]
y_combined_4 = final_dataframe['normalized_score']
```

```
y_true = np.array(y_valid_4)
y_pred = np.array(predictions_4)
y_true = np.round(y_true*10).astype(int)
y_pred = np.round(y_pred*10).astype(int)


print(compute_qwk(y_true, y_pred))
```

```
0.5486859519757987
```

```
y_true = np.array(y_valid)
y_pred = np.array(y_pred)
count=0
for i in range(len(y_pred)):
    print(f"Predicted Score : {y_pred[i]} -> Real Score :  {np.round(y_true[i]*10)}")
```

```
Predicted Score : 5 -> Real Score :  5.0
Predicted Score : 5 -> Real Score :  4.0
Predicted Score : 7 -> Real Score :  7.0
Predicted Score : 5 -> Real Score :  5.0
Predicted Score : 8 -> Real Score :  8.0
Predicted Score : 7 -> Real Score :  7.0
Predicted Score : 6 -> Real Score :  6.0
Predicted Score : 5 -> Real Score :  6.0
Predicted Score : 7 -> Real Score :  7.0
Predicted Score : 7 -> Real Score :  8.0
Predicted Score : 6 -> Real Score :  6.0
Predicted Score : 7 -> Real Score :  8.0
Predicted Score : 5 -> Real Score :  4.0
Predicted Score : 6 -> Real Score :  6.0
Predicted Score : 5 -> Real Score :  6.0
Predicted Score : 5 -> Real Score :  4.0
Predicted Score : 4 -> Real Score :  4.0
Predicted Score : 6 -> Real Score :  6.0
Predicted Score : 5 -> Real Score :  5.0
Predicted Score : 6 -> Real Score :  7.0
Predicted Score : 5 -> Real Score :  5.0
Predicted Score : 7 -> Real Score :  6.0
Predicted Score : 6 -> Real Score :  7.0
Predicted Score : 8 -> Real Score :  8.0
Predicted Score : 7 -> Real Score :  6.0
Predicted Score : 8 -> Real Score :  8.0
```

SECOND COMBINED MODEL

For second model we use semantic score novelty and coherence score novelity

COMBINED MODEL 3

```
X_combined_3 = final_dataframe[[ 'out_semantic', 'out_coher', 'out_prompt', 'combined_score', 'coherence_s
y_combined_3 = final_dataframe['normalized_score']
```

```
50]: y_true = np.array(y_valid_3)
     y_pred = np.array(predictions_3)
     y_true = np.round(y_true*10).astype(int)
     y_pred = np.round(y_pred*10).astype(int)


     print(compute_qwk(y_true, y_pred))
```

```
0.5486859519757987
```

We got a qwk score of 0.55 approximately.

THIRD COMBINED MODEL

For this model we used coherence score novelty combing with handcrafted features and ranks novelity

COMBINED MODEL 2

```
import sklearn
from sklearn.model_selection import train_test_split
X_combined_2 = final_dataframe[['means', 'variances', 'grammaticals', 'word_counts', 'char_counts', 'out_cohe
y_combined_2 = final_dataframe['normalized_score']
```

```
y_true = np.array(y_valid_2)
y_pred = np.array(predictions_2)
y_true = np.round(y_true*10).astype(int)
y_pred = np.round(y_pred*10).astype(int)


print(compute_qwk(y_true, y_pred))
```

```
0.49744998038446453
```

BEST MODEL

In our best model we combined previous score along with all our 3 novelities

```
import sklearn
from sklearn.model_selection import train_test_split
# means variances grammaticals word_counts char_counts out_semantic out_coher out_prompt combined_score
X = main_data[['means', 'variances', 'grammaticals', 'word_counts', 'char_counts', 'out_semantic',
               'out_coher', 'out_prompt', 'combined_score', 'coherence_score_nli','ranks']]
y = main_data['normalized_score']

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Example usage
y_true = np.array(y_valid)
y_pred = np.array(y_pred)
y_true = np.round(y_true*10).astype(int)
y_pred = np.round(y_pred*10).astype(int)


print(compute_qwk(y_true, y_pred))
```

```
0.6238198825320863
```

```
print(count, len(y_pred))
```

```
Predicted Score : 7 -> Real Score :  8.0
Predicted Score : 7 -> Real Score :  7.0
Predicted Score : 5 -> Real Score :  5.0
Predicted Score : 7 -> Real Score :  8.0
Predicted Score : 3 -> Real Score :  2.0
Predicted Score : 7 -> Real Score :  8.0
Predicted Score : 5 -> Real Score :  4.0
Predicted Score : 3 -> Real Score :  3.0
Predicted Score : 6 -> Real Score :  7.0
Predicted Score : 5 -> Real Score :  6.0
Predicted Score : 7 -> Real Score :  6.0
Predicted Score : 6 -> Real Score :  7.0
Predicted Score : 6 -> Real Score :  5.0
Predicted Score : 5 -> Real Score :  4.0
Predicted Score : 7 -> Real Score :  8.0
Predicted Score : 4 -> Real Score :  3.0
Predicted Score : 5 -> Real Score :  6.0
Predicted Score : 5 -> Real Score :  4.0
Predicted Score : 5 -> Real Score :  6.0
```

We got qwk score of 0.624

WEBSITE DESIGN:

We actually designed a small website in which we enter the prompt and essay we will get the score.

We are actually using our best pretrained model for generating output.

Automated Essay Scoring

Prompt:

words @CAPS1, Narciso knows even though they only had a three - room apartment he knew it was special. Narciso states, "I will never forget how my parents turned this simple house into a home" (@NUM3). His parents couldnt give him much because they didn't have much money but it meant the world to him Clearly, the mood of this passage is gratitude

Essay Text:

In the memoir, Narciso Rodriguez, by Narciso Rodriguez the mood created by the author is gratitude. First, Narciso loved his parents and was very grateful for everything they did. Rodriguez says, "I will always be grateful to my parents for their love and sacrafice" (@NUM1). Narciso's parents came to the US just for him and he is so grateful. Secondly, he says he has

Predict

Predicted Score: 4

4. Implementation Plan:

l'ask No.	Task	Estimated Completion Date
1	Project Outline	11/09/2023
2	Data Preprocessing	13/09/2023 - 18/09/2023
3	Interim Submission	12/10/2023 - 18/10/2023
4	Model Creation	12/10/2023 - 18/10/2023
5	Cross Analysis and Examination	15/11/2023 - 18/11/2023
6	Final Report Submission	17/11/2023 - 20/11/2023

5. Progress and challenges:

Both baseline and novelity is successfully implemented.

For baseline our qwk score was less 0.42 but for novelty we got a qwk score off 0.62 which is decent enough as the original paper has qwk score close to that.

The main issue we faced was in training a mode because of limited GPU and computation power we were not able to train few models on large dataset.
We had toreduce our dataset thats why we could cross the qwk score of actual paper but we were close to that.

We have faced the following challenges during the implementation of baseline:

1) Large Dataset Sizes: Training AES models often requires substantial amounts of training data, and processing large datasets can be time-consuming.
2) Model Complexity: Due to the architecture and depth of the model, the training and scoring processes can be computationally intensive, leading to longer processing times. It is computationally expensive as we are making embeddings of each word and sentence in an essay.
3) For each model, we have to create separate dataframes, which is a time consuming task.

References:

1. Automated Essay Scoring based on Two-Stage Learning Jiawei Liu† , Yang Xu‡ , and Yaguang Zhu†
2. A Neural Approach to Automated Essay Scoring Kaveh Taghipour and Hwee Tou Ng
3. Joint Learning of Multi-Scale Essay Representation Yongjie Wang1 Chuan Wang1 Ruobing Li1 Hui Lin1,2
4. Countering the Influence of Essay Length in Neural Essay Scoring Sungho Jeon and Michael Strube
5. Flexible Domain Adaptation for Automated Essay Scoring Using Correlated Linear Regression Peter Phandi1 Kian Ming A. Chai2 Hwee Tou Ng1
6. https://medium.com/red-buffer/natural-language-inference-using-bert-and-pytorch-6ed8e69f93bc
7. Automated Essay Scoring via Pairwise Contrastive Regression-Jiayi Xie∗ , Kaiwei Cai∗ , Li Kong, Junsheng Zhou†, Weiguang Qu
8. https://www.sciencedirect.com/science/article/pii/S266638642300200X?ref=pdf_download&fr=RR-2&rr=829200826e3df2f3