

Learning Agency Lab - Automated Essay Scoring 2.0

Learning Agency Lab - Automated Essay Scoring 2.0:- The goal is to build a model that can accurately predict the score an essay deserves based solely on its text content. The competition aims to improve student learning outcomes by providing timely and reliable feedback to overburdened educators.

Problem Statement

Essay writing is a crucial method to evaluate student learning and performance, but it is time-consuming for educators to grade manually.

- **Automated Writing Evaluation (AWE)** systems can assist in scoring essays, providing students with regular and timely feedback. However, many advancements in AWE are not widely accessible due to cost barriers. Open-source solutions are needed to make AWE technology available to every community, especially underserved ones.

Competition Objective

The objective of this competition is to train a model to score student essays accurately. Participants are tasked with reducing the high expense and time required for manual grading, making it feasible to introduce essays into testing, a key indicator of student learning.

Dataset

The competition dataset comprises about 24000 student-written argumentative essays. Each essay was scored on a scale of 1 to 6 (Link to the Holistic Scoring Rubric). Your goal is to predict the score an essay received from its text.

File and Field Information:- Sure, here's the information organized in a tabular form:

File Name	Description	Fields
train.csv	Essays and scores to be used as training data	essay_id, full_text, score
test.csv	Essays to be used as test data	essay_id, full_text
sample_submission.csv	A submission file in the correct format	essay_id, score

Each file contains specific fields:

- `train.csv` : Contains essays along with their unique ID (`essay_id`), the full text of the essay (`full_text`), and the holistic score of the essay on a 1-6 scale (`score`).
- `test.csv` : Contains essays to be used as test data, including their unique ID (`essay_id`) and the full text of the essay (`full_text`). This file does not include the `score` field.
- `sample_submission.csv` : A submission file template with the correct format for submission. It includes the unique ID of each essay (`essay_id`) and a placeholder for the predicted holistic score of the essay on a 1-6 scale (`score`).

This tabular representation summarizes the contents of each file and their respective fields, providing clarity on the dataset structure and file formats.

Evaluation

Submissions are scored based on the quadratic weighted kappa, which measures the agreement between two outcomes. This metric typically varies from 0 (random agreement) to 1 (complete agreement). In the event that there is less agreement than expected by chance, the metric may go below 0.

The quadratic weighted kappa is calculated as follows. First, an $N \times N$ histogram matrix O is constructed, such that $O_{i,j}$ corresponds to the number of essay_ids i (actual) that received a predicted value j . An N -by- N matrix of weights, w , is calculated based on the difference between actual and predicted values:

$$w_{i,j} = \frac{(i-j)^2}{(N-1)^2}$$

An N -by- N histogram matrix of expected outcomes, E , is calculated assuming that there is no correlation between values. This is calculated as the outer product between the actual histogram vector of outcomes and the predicted histogram vector, normalized such that E and O have the same sum.

From these three matrices, the quadratic weighted kappa is calculated as:

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}.$$

Submission File

For each `essay_id` in the test set, participants must predict the corresponding score. The submission file should contain a header and have the following format:

```
essay_id,score
000d118,3
000fe60,3
001ab80,4
```

For detailed instructions, guidelines, and access to the dataset, please visit the competition page on Kaggle: [Learning Agency Lab - Automated Essay Scoring 2.0](#)

1. Import modules

```
In [ ]: # !pip install "/kaggle/input/pyspellchecker/pyspellchecker-0.7.2-py3-none-any.w
```

```
In [ ]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import polars as pl

from sklearn.model_selection import StratifiedKFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import cohen_kappa_score
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize.treebank import TreebankWordDetokenizer

import string
import re
from spellchecker import SpellChecker
import lightgbm as lgb

import warnings
import logging
warnings.simplefilter("ignore")
logging.disable(logging.ERROR)
```

2. Load dataset and initial configuration

```
In [ ]: class PATHS:
    train_path = 'data/train.csv'
    test_path = 'data/test.csv'
    sub_path = 'data/sample_submission.csv'
```

```
In [ ]: class CFG:
    n_splits = 5
    seed = 42
    num_labels = 6
```

```
In [ ]: train = pd.read_csv(PATHS.train_path)
train.head(3)
```

```
Out[ ]:
```

	essay_id	full_text	score
0	000d118	Many people have car where they live. The thin...	3
1	000fe60	I am a scientist at NASA that is discussing th...	3
2	001ab80	People always wish they had the same technolog...	4

```
In [ ]: test = pd.read_csv(PATHS.test_path)
test.head(3)
```

```
Out[ ]:      essay_id      full_text
0  000d118  Many people have car where they live. The thin...
1  000fe60  I am a scientist at NASA that is discussing th...
2  001ab80  People always wish they had the same technolog...
```

3. Feature Engineering

3.1 Data preprocessing functions definations

```
In [ ]: def removeHTML(x):
    html=re.compile(r'<.*?>')
    return html.sub(r'',x)

cList = {
    "ain't": "am not", "aren't": "are not", "can't": "cannot", "can't've": "cann",
    "couldn't": "could not", "couldn't've": "could not have", "didn't": "did not",
    "hadn't've": "had not have", "hasn't": "has not", "haven't": "have not",
    "he'd": "he would", ## --> he had or he would
    "he'd've": "he would have", "he'll": "he will", "he'll've": "he will have", "
    "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's": "ho",
    "I'd": "I would", ## --> I had or I would
    "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have", "I'm": "I",
    "it'd": "it had", ## --> It had or It would
    "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have", "it",
    "let's": "let us", "ma'am": "madam", "mayn't": "may not", "might've": "might ha",
    "must've": "must have", "mustn't": "must not", "mustn't've": "must not have",
    "needn't": "need not", "needn't've": "need not have",
    "o'clock": "of the clock",
    "oughtn't": "ought not", "oughtn't've": "ought not have",
    "shan't": "shall not", "sha'n't": "shall not", "shan't've": "shall not have",
    "she'd": "she would", ## --> It had or It would
    "she'd've": "she would have", "she'll": "she will", "she'll've": "she will hav",
    "should've": "should have", "shouldn't": "should not", "shouldn't've": "should",
    "so've": "so have", "so's": "so is",
    "that'd": "that would",
    "that'd've": "that would have", "that's": "that is",
    "there'd": "there had",
    "there'd've": "there would have", "there's": "there is",
    "they'd": "they would",
    "they'd've": "they would have", "they'll": "they will", "they'll've": "they wi",
    "to've": "to have", "wasn't": "was not", "weren't": "were not",
    "we'd": "we had",
    "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we",
    "what'll": "what will", "what'll've": "what will have", "what're": "what are",
    "when's": "when is", "when've": "when have",
    "where'd": "where did", "where's": "where is", "where've": "where have",
    "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've",
    "will've": "will have", "won't": "will not", "won't've": "will not have",
```

```

    "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not h
    'y'all": "you all", "y'all's": "you alls", "y'all'd": "you all would", "y'all'd'
    'y'all've": "you all have", "you'd": "you had", "you'd've": "you would have", "
    'you're": "you are", "you've": "you have"
}
c_re = re.compile('(' + '%s)' % '|'.join(cList.keys()))

def expandContractions(text):
    def replace(match):
        return cList[match.group(0)]
    return c_re.sub(replace, text)

def dataPreprocessing(x):
    # Convert words to lowercase
    x = x.lower()
    # Remove HTML
    x = removeHTML(x)
    # Delete strings starting with @
    x = re.sub("@\w+", '', x)
    # Delete Numbers
    x = re.sub("\d+", '', x)
    x = re.sub("\d+", '', x)
    # Delete URL
    x = re.sub("http\w+", '', x)
    # Remove \xa0
    x = x.replace(u'\xa0', ' ')
    # Replace consecutive empty spaces with a single space character
    x = re.sub(r"\s+", " ", x)
    x = expandContractions(x)
    # Replace consecutive commas and periods with one comma and period character
    x = re.sub(r"\.+", ".", x)
    x = re.sub(r"\,+", ",", x)
    # Remove empty characters at the beginning and end
    x = x.strip()
    return x

def remove_punctuation(text):
    # string.punctuation
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

def dataPreprocessing_w_contract_punct_remove(x):
    # Convert words to lowercase
    x = x.lower()
    # Remove HTML
    x = removeHTML(x)
    # Delete strings starting with @
    x = re.sub("@\w+", '', x)
    # Delete Numbers
    x = re.sub("\d+", '', x)
    x = re.sub("\d+", '', x)
    # Delete URL
    x = re.sub("http\w+", '', x)
    # Replace consecutive empty spaces with a single space character
    x = re.sub(r"\s+", " ", x)
    x = expandContractions(x)
    # Replace consecutive commas and periods with one comma and period character
    x = re.sub(r"\.+", ".", x)
    x = re.sub(r"\,+", ",", x)
    x = remove_punctuation(x)

```

```
# Remove empty characters at the beginning and end
x = x.strip()
return x
```

3.2 Paragraph based feature

```
In [ ]: # TODO: can be fixed by keeping "\n" and removed empty paragraph entries
columns = [(pl.col("full_text").str.split(by="\n\n").alias("paragraph"))]
train = pl.from_pandas(train).with_columns(columns)
test = pl.from_pandas(test).with_columns(columns)
```

```
In [ ]: # paragraph features
def Paragraph_Preprocess(tmp):
    # Expand the paragraph list into several lines of data
    tmp = tmp.explode('paragraph')
    # Paragraph preprocessing
    tmp = tmp.with_columns(pl.col('paragraph').map_elements(dataPreprocessing))
    # Calculate the length of each paragraph
    tmp = tmp.with_columns(pl.col('paragraph').map_elements(lambda x: len(x)).alias('paragraph_len'))
    # Calculate the number of sentences and words in each paragraph
    tmp = tmp.with_columns(pl.col('paragraph').map_elements(lambda x: len(x.split('. '))).alias('paragraph_sentence_cnt'))
    tmp = tmp.with_columns(pl.col('paragraph').map_elements(lambda x: len(x.split(' '))).alias('paragraph_word_cnt'))
    return tmp

# feature_eng
paragraph_fea = ['paragraph_len', 'paragraph_sentence_cnt', 'paragraph_word_cnt']
def Paragraph_Eng(train_tmp):
    aggs = [
        # Count the number of paragraph lengths greater than and less than the i
        *[pl.col('paragraph').filter(pl.col('paragraph_len') >= i).count().alias(f'count_{i}_greater_than_or_equal'),
          pl.col('paragraph').filter(pl.col('paragraph_len') <= i).count().alias(f'count_{i}_less_than_or_equal')
        for i in range(1, 1000)
    ]
    # other
    *[(pl.col(fea).max().alias(f'{fea}_max') for fea in paragraph_fea),
      (pl.col(fea).mean().alias(f'{fea}_mean') for fea in paragraph_fea),
      (pl.col(fea).min().alias(f'{fea}_min') for fea in paragraph_fea),
      (pl.col(fea).first().alias(f'{fea}_first') for fea in paragraph_fea),
      (pl.col(fea).last().alias(f'{fea}_last') for fea in paragraph_fea),
      (pl.col(fea).sum().alias(f'{fea}_sum') for fea in paragraph_fea),
      (pl.col(fea).kurtosis().alias(f'{fea}_kurtosis') for fea in paragraph_fea),
      (pl.col(fea).quantile(0.25).alias(f'{fea}_q1') for fea in paragraph_fea),
      (pl.col(fea).quantile(0.75).alias(f'{fea}_q3') for fea in paragraph_fea)
    ]
    df = train_tmp.group_by(['essay_id'], maintain_order=True).agg(aggs).sort("essay_id")
    df = df.to_pandas()
    return df

tmp = Paragraph_Preprocess(train)
train_feats = Paragraph_Eng(tmp)

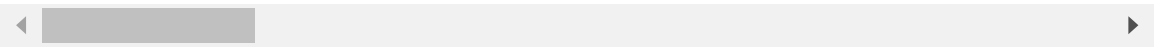
# Obtain feature names
feature_names = list(filter(lambda x: x not in ['essay_id', 'score'], train_feats.columns))
print('Features Number: ', len(feature_names))
train_feats.head(5)
```

Features Number: 43

Out[]:

	essay_id	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt
0	000d118	1	1	1	1
1	000fe60	5	5	5	5
2	001ab80	4	4	4	4
3	001bdc0	5	5	5	5
4	002ba53	4	4	4	4

5 rows × 44 columns



3.3 Sentence based features

source: <https://www.kaggle.com/code/ye11725/tfidf-lgbm-baseline-with-code-comments/notebook#Features-engineering>

```
In [ ]: # sentence feature
def Sentence_Preprocess(tmp):
    # Preprocess full_text and use periods to segment sentences in the text
    tmp = tmp.with_columns(pl.col('full_text').map_elements(dataPreprocessing).s
    tmp = tmp.explode('sentence')
    # Calculate the length of a sentence
    tmp = tmp.with_columns(pl.col('sentence').map_elements(lambda x: len(x)).ali
    # Filter out the portion of data with a sentence length greater than 15
    tmp = tmp.filter(pl.col('sentence_len')>=15)
    # Count the number of words in each sentence
    tmp = tmp.with_columns(pl.col('sentence').map_elements(lambda x: len(x.split
    return tmp

# feature_eng
sentence_fea = ['sentence_len', 'sentence_word_cnt']
def Sentence_Eng(train_tmp):
    aggs = [
        # Count the number of sentences with a length greater than i
        *[pl.col('sentence').filter(pl.col('sentence_len') >= i).count().alias(f
        # other
        *[pl.col(fea).max().alias(f"{fea}_max") for fea in sentence_fea],
        *[pl.col(fea).mean().alias(f"{fea}_mean") for fea in sentence_fea],
        *[pl.col(fea).min().alias(f"{fea}_min") for fea in sentence_fea],
        *[pl.col(fea).first().alias(f"{fea}_first") for fea in sentence_fea],
        *[pl.col(fea).last().alias(f"{fea}_last") for fea in sentence_fea],
        *[pl.col(fea).sum().alias(f"{fea}_sum") for fea in sentence_fea],
        *[pl.col(fea).kurtosis().alias(f"{fea}_kurtosis") for fea in sentence_fe
        *[pl.col(fea).quantile(0.25).alias(f"{fea}_q1") for fea in sentence_fea]
        *[pl.col(fea).quantile(0.75).alias(f"{fea}_q3") for fea in sentence_fea]
    ]
    df = train_tmp.group_by(['essay_id'], maintain_order=True).agg(aggs).sort("e
    df = df.to_pandas()
    return df

tmp = Sentence_Preprocess(train)

# Merge the newly generated feature data with the previously generated feature d
```

```
train_feats = train_feats.merge(Sentence_Eng(tmp), on='essay_id', how='left')

feature_names = list(filter(lambda x: x not in ['essay_id', 'score'], train_feats
print('Features Number: ', len(feature_names))
train_feats.head(3)
```

Features Number: 68

Out[]:

	essay_id	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt
0	000d118	1	1	1	1
1	000fe60	5	5	5	5
2	001ab80	4	4	4	4

3 rows × 69 columns

3.4 Word based feature

source: <https://www.kaggle.com/code/ye11725/tfidf-lgbm-baseline-with-code-comments/notebook#Features-engineering>

```
In [ ]: # word feature
def Word_Preprocess(tmp):
    # Preprocess full_text and use spaces to separate words from the text
    tmp = tmp.with_columns(pl.col('full_text').map_elements(dataPreprocessing).s
    tmp = tmp.explode('word')
    # Calculate the length of each word
    tmp = tmp.with_columns(pl.col('word').map_elements(lambda x: len(x)).alias("
    # Delete data with a word length of 0
    tmp = tmp.filter(pl.col('word_len')!=0)

    return tmp

# feature_eng
def Word_Eng(train_tmp):
    aggs = [
        # Count the number of words with a length greater than i+1
        *[pl.col('word').filter(pl.col('word_len') >= i+1).count().alias(f"word_
        # other
        pl.col('word_len').max().alias(f"word_len_max"),
        pl.col('word_len').mean().alias(f"word_len_mean"),
        pl.col('word_len').std().alias(f"word_len_std"),
        pl.col('word_len').quantile(0.25).alias(f"word_len_q1"),
        pl.col('word_len').quantile(0.50).alias(f"word_len_q2"),
        pl.col('word_len').quantile(0.75).alias(f"word_len_q3"),
    ]
    df = train_tmp.group_by(['essay_id'], maintain_order=True).agg(aggs).sort("e
    df = df.to_pandas()
    return df

tmp = Word_Preprocess(train)

# Merge the newly generated feature data with the previously generated feature d
train_feats = train_feats.merge(Word_Eng(tmp), on='essay_id', how='left')
```



```
feature_names = list(filter(lambda x: x not in ['essay_id', 'score'], train_feats
print('Features Number: ', len(feature_names))
train_feats.head(3)
```

Features Number: 89

```
Out[ ]:      essay_id  paragraph_50_cnt  paragraph_75_cnt  paragraph_100_cnt  paragraph_125_cnt
0  000d118              1              1              1              1
1  000fe60              5              5              5              5
2  001ab80              4              4              4              4
```

3 rows × 90 columns



3.5 Character TFIDF feature:

For TFIDF vector generation we use TfidfVectorizer provided by [sickit-learn](#) library

Terms:

- **TF (Term frequency):** Number of time a term occur in a document / Total number of term in the document.
- **DF (Document frequency):** Number of document where the term appear / Total number of document.
- **IDF (Inverse Document Frequency):** 1 / Document frequency

TfidfVectorizer parameters:

- **tokenizer:** Is set to `lambda x: x` which means the text will be passed as it is.
- **preprocessor:** Is set to `lambda x: x` which means the text will be passed as it is.
- **token_pattern:** Is not set to `None` means word will be taken as token as it is without any word-level processing.
- **strip_accents:** Is set to `unicode` which means include unicode characters during preprocessing step.
- **analyzer:** Is set to `word` which means the feature (terms or token) will be the words
- **ngram_range:** ngram_range equal to `(1, 2)` which means unigrams and bigrams
- **min_df:** Is equal to `0.05` means ignore terms that occur in less the 5% of documents.
- **max_df:** Is equal to `0.95` means ignore terms that occur in more them 95% of documents.
- **sublinear_tf:** Is equal to `True` means replace tf with $1 + \log(\text{tf})$

Note:

- **tokenizer=lambda x: x:** " words are not tokenized from full-text?
Tokenizer should only be overided by identity if text is already tokenized before. Perhaps vectorizer is receiving string (char

sequence) instead of word sequence, so it behaves like a char ngram vectorizer " quoted from notebook [here](#)

```
In [ ]: # TfidfVectorizer parameter
vectorizer = TfidfVectorizer(
    tokenizer=lambda x: x,
    preprocessor=lambda x: x,
    token_pattern=None,
    strip_accents='unicode',
    analyzer = 'word',
    ngram_range=(1,3),
    min_df=0.05,
    max_df=0.95,
    sublinear_tf=True,
)
# Fit all datasets into TfidfVector, this may cause leakage and overly optimistic
train_tfidf = vectorizer.fit_transform([i for i in train['full_text']])

print("#"*80)
vect_feat_names=vectorizer.get_feature_names_out()
print(vect_feat_names[100:110])
print("#"*80, "\n\n")

# Convert to array
dense_matrix = train_tfidf.toarray()

# Convert to dataframe
df = pd.DataFrame(dense_matrix)

# rename features
tfidf_columns = [ f'tfidf_{i}' for i in range(len(df.columns))]
df.columns = tfidf_columns
df['essay_id'] = train_feats['essay_id']

# Merge the newly generated feature data with the previously generated feature d
train_feats = train_feats.merge(df, on='essay_id', how='left')

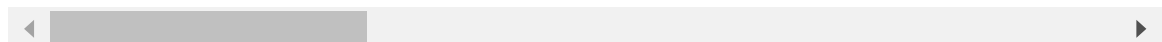
feature_names = list(filter(lambda x: x not in ['essay_id','score'], train_feats
print('Features Number: ',len(feature_names))
train_feats.head(3)
```

```
#####
[' D e ' ' D o ' ' D r ' ' E ' ' E a ' ' E l ' ' E u ' ' E v ' ' E x '
 ' F']
#####
```

Features Number: 3380

```
Out[ ]:   essay_id  paragraph_50_cnt  paragraph_75_cnt  paragraph_100_cnt  paragraph_125_cnt
0  000d118                1                1                1                1
1  000fe60                5                5                5                5
2  001ab80                4                4                4                4
```

3 rows × 3381 columns



```
In [ ]: stopwords_list = stopwords.words('english')
# TfidfVectorizer parameter
word_vectorizer = TfidfVectorizer(
    strip_accents='ascii',
    analyzer = 'word',
    ngram_range=(1,1),
    min_df=0.05,
    max_df=0.95,
    sublinear_tf=True,
    stop_words=stopwords_list,
)
# Fit all datasets into TfidfVectorizer, this may cause leakage and overly optimistic
processed_text = train.to_pandas()["full_text"].apply(lambda x: dataPreprocessin
train_tfidf = word_vectorizer.fit_transform([i for i in processed_text])

# Convert to array
dense_matrix = train_tfidf.toarray()
# Convert to dataframe
df = pd.DataFrame(dense_matrix)
# rename features
tfidf_w_columns = [ f'tfidf_w_{i}' for i in range(len(df.columns))]
df.columns = tfidf_w_columns
df['essay_id'] = train_feats['essay_id']

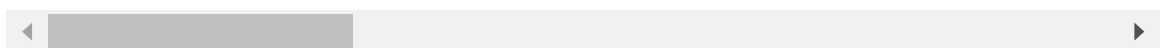
df.head()
# Merge the newly generated feature data with the previously generated feature d
train_feats = train_feats.merge(df, on='essay_id', how='left')

feature_names = list(filter(lambda x: x not in ['essay_id', 'score'], train_feats
print('Features Number: ', len(feature_names))
train_feats.head(3)
```

Features Number: 3894

```
Out [ ]:      essay_id  paragraph_50_cnt  paragraph_75_cnt  paragraph_100_cnt  paragraph_125_cnt
0  000d118                1                1                1                1
1  000fe60                5                5                5                5
2  001ab80                4                4                4                4
```

3 rows × 3895 columns



3.7 Extra features:

Reference: <https://www.kaggle.com/code/tsunotsuno/updated-debertav3-lgbm-with-spell-autocorrect>

```
In [ ]: class Preprocessor:
    def __init__(self) -> None:
        self.twd = TreebankWordDetokenizer()
        self.STOP_WORDS = set(stopwords.words('english'))
        self.spellchecker = SpellChecker()

    def spelling(self, text):
```

```

wordlist=text.split()
amount_miss = len(list(self.spellchecker.unknown(wordlist)))
return amount_miss

def count_sym(self, text, sym):
    sym_count = 0
    for l in text:
        if l == sym:
            sym_count += 1
    return sym_count

def run(self, data: pd.DataFrame, mode:str) -> pd.DataFrame:

    # preprocessing the text
    data["processed_text"] = data["full_text"].apply(lambda x: dataPreproces

    # Text tokenization
    data["text_tokens"] = data["processed_text"].apply(lambda x: word_tokeni

    # essay length
    data["text_length"] = data["processed_text"].apply(lambda x: len(x))

    # essay word count
    data["word_count"] = data["text_tokens"].apply(lambda x: len(x))

    # essay unique word count
    data["unique_word_count"] = data["text_tokens"].apply(lambda x: len(set(

    # essay sentence count
    data["sentence_count"] = data["full_text"].apply(lambda x: len(x.split('

    # essay paragraph count
    data["paragraph_count"] = data["full_text"].apply(lambda x: len(x.split(

    # count misspelling
    data["spelling_err_num"] = data["processed_text"].apply(self.spelling)
    print("Spelling mistake count done")

    return data

```

```

In [ ]: preprocessor = Preprocessor()
tmp = preprocessor.run(train.to_pandas(), mode="train")
train_feats = train_feats.merge(tmp, on='essay_id', how='left')
feature_names = list(filter(lambda x: x not in ['essay_id', 'score'], train_feats

```

Spelling mistake count done

```

In [ ]: print('Features Number: ',len(feature_names))
train_feats.head(3)

```

Features Number: 3904

Out []:

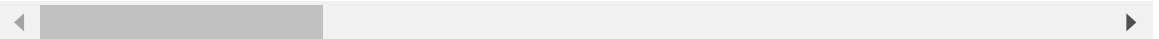
	essay_id	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt
--	----------	------------------	------------------	-------------------	-------------------

0	000d118	1	1	1	1
---	---------	---	---	---	---

1	000fe60	5	5	5	5
---	---------	---	---	---	---

2	001ab80	4	4	4	4
---	---------	---	---	---	---

3 rows × 3906 columns



3.8 Test dataset featurization

```
In [ ]: # Paragraph
tmp = Paragraph_Preprocess(test)
test_feats = Paragraph_Eng(tmp)

# Sentence
tmp = Sentence_Preprocess(test)
test_feats = test_feats.merge(Sentence_Eng(tmp), on='essay_id', how='left')

# Word
tmp = Word_Preprocess(test)
test_feats = test_feats.merge(Word_Eng(tmp), on='essay_id', how='left')
```

```
In [ ]: # Tfidf
test_tfidf = vectorizer.transform([i for i in test['full_text']])
dense_matrix = test_tfidf.toarray()
df = pd.DataFrame(dense_matrix)
tfidf_columns = [f'tfidf_{i}' for i in range(len(df.columns))]
df.columns = tfidf_columns
df['essay_id'] = test_feats['essay_id']
test_feats = test_feats.merge(df, on='essay_id', how='left')
```

```
In [ ]: # Word Tfidf
processed_text = test.to_pandas()["full_text"].apply(lambda x: dataPreprocessing)
# train_w_tfidf = word_vectorizer.fit_transform(train['full_text'])
test_w_tfidf = word_vectorizer.fit_transform([i for i in processed_text])
dense_matrix = test_w_tfidf.toarray()
df_w = pd.DataFrame(dense_matrix)
tfidf_w_columns = [f'tfidf_w_{i}' for i in range(len(df_w.columns))]
df_w.columns = tfidf_w_columns
```

```
df_w['essay_id'] = test_feats['essay_id']
test_feats = test_feats.merge(df_w, on='essay_id', how='left')
```

In []: *# Extra feature*

```
preprocessor2 = Preprocessor()
tmp = preprocessor2.run(test.to_pandas(), mode="train")
test_feats = test_feats.merge(tmp, on='essay_id', how='left')
```

Spelling mistake count done

In []: test_feats.head(3)

Out []:

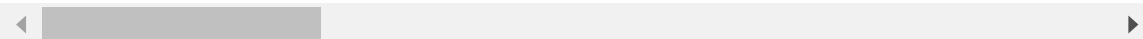
	essay_id	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt
--	----------	------------------	------------------	-------------------	-------------------

0	000d118	1	1	1	1
---	---------	---	---	---	---

1	000fe60	5	5	5	5
---	---------	---	---	---	---

2	001ab80	4	4	4	4
---	---------	---	---	---	---

3 rows × 3759 columns



In []: *# Features number*

```
feature_names = list(filter(lambda x: x not in ['essay_id', 'score'], test_feats.
print('Features number: ', len(feature_names))
test_feats.head(3))
```

Features number: 3758

Out []:

	essay_id	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt
0	000d118	1	1	1	1
1	000fe60	5	5	5	5
2	001ab80	4	4	4	4

3 rows × 3759 columns

4. Data preparation

4.1 Add k-fold details

In []:

```
skf = StratifiedKFold(n_splits=CFG.n_splits, shuffle=True, random_state=CFG.seed)
for i, (_, val_index) in enumerate(skf.split(train_feats, train_feats["score"])):
    train_feats.loc[val_index, "fold"] = i
print(train_feats.shape)
# train_feats.head()
```

(17307, 3907)

In []:

```
test_feats.shape
```

Out []: (3, 3759)

4.2 Feature selection

In []:

```
target = "score"
train_drop_columns = ["essay_id", "fold", "full_text", "paragraph", "text_tokens"]
```

In []:

```
train_feats.drop(columns=train_drop_columns).head()
```

Out []:

	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt	paragra
0	1	1	1	1	
1	5	5	5	5	
2	4	4	4	4	
3	5	5	5	5	
4	4	4	4	4	

5 rows × 3754 columns



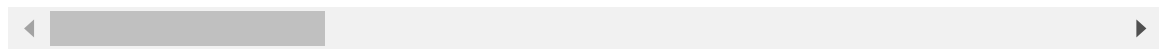
In []: `test_drop_columns = ["essay_id", "full_text", "paragraph", "text_tokens", "proce`

In []: `test_feats.drop(columns=test_drop_columns).head()`

Out []:

	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt	paragra
0	1	1	1	1	
1	5	5	5	5	
2	4	4	4	4	

3 rows × 3754 columns



5. Training

5.1 Evaluation function and loss function defination

```
In [ ]: # idea from https://www.kaggle.com/code/rsakata/optimize-qwk-by-Lgb/notebook#QWK
def quadratic_weighted_kappa(y_true, y_pred):
    y_true = y_true + a
    y_pred = (y_pred + a).clip(1, 6).round()
    qwk = cohen_kappa_score(y_true, y_pred, weights="quadratic")
    return 'QWK', qwk, True

def qwk_obj(y_true, y_pred):
    labels = y_true + a
    preds = y_pred + a
    preds = preds.clip(1, 6)
    f = 1/2*np.sum((preds-labels)**2)
    g = 1/2*np.sum((preds-a)**2+b)
    df = preds - labels
    dg = preds - a
    grad = (df/g - f*dg/g**2)*len(labels)
    hess = np.ones(len(labels))
    return grad, hess
```



```
a = 2.948
b = 1.092
```

5.2 Training LGBMRegressor model

```
In [ ]: models = []

callbacks = [
    lgb.log_evaluation(period=25),
    lgb.early_stopping(stopping_rounds=75, first_metric_only=True)
]
for fold in range(CFG.n_splits):

    model = lgb.LGBMRegressor(
        objective = qwk_obj, metrics = 'None', learning_rate = 0.1, max_depth =
        num_leaves = 10, colsample_bytree=0.5, reg_alpha = 0.1, reg_lambda = 0.8
        n_estimators=1024, random_state=CFG.seed, verbosity = - 1
    )

    # Take out the training and validation sets for 5 kfold segmentation separat
    X_train = train_feats[train_feats["fold"] != fold].drop(columns=train_drop_c
    y_train = train_feats[train_feats["fold"] != fold]["score"] - a

    X_eval = train_feats[train_feats["fold"] == fold].drop(columns=train_drop_co
    y_eval = train_feats[train_feats["fold"] == fold]["score"] - a

    print('\nFold_{0} Training =====\n'.format(fold+1))
    # Training model
    lgb_model = model.fit(
        X_train, y_train,
        eval_names=['train', 'valid'],
        eval_set=[(X_train, y_train), (X_eval, y_eval)],
        eval_metric=quadratic_weighted_kappa,
        callbacks=callbacks
    )
    models.append(model)
```

Fold_1 Training =====

```
[LightGBM] [Info] Using self-defined objective function
Training until validation scores don't improve for 75 rounds
[25]   train's QWK: 0.755278   valid's QWK: 0.744537
[50]   train's QWK: 0.800207   valid's QWK: 0.777743
[75]   train's QWK: 0.817787   valid's QWK: 0.788816
[100]  train's QWK: 0.828686   valid's QWK: 0.793632
[125]  train's QWK: 0.83737    valid's QWK: 0.795988
[150]  train's QWK: 0.844961   valid's QWK: 0.796691
[175]  train's QWK: 0.851675   valid's QWK: 0.801721
[200]  train's QWK: 0.858156   valid's QWK: 0.800466
[225]  train's QWK: 0.863782   valid's QWK: 0.798881
[250]  train's QWK: 0.869761   valid's QWK: 0.799522
Early stopping, best iteration is:
[181]  train's QWK: 0.853508   valid's QWK: 0.802415
Evaluated only: QWK
```

Fold_2 Training =====

```
[LightGBM] [Info] Using self-defined objective function
Training until validation scores don't improve for 75 rounds
[25]   train's QWK: 0.752572   valid's QWK: 0.74094
[50]   train's QWK: 0.79462    valid's QWK: 0.786217
[75]   train's QWK: 0.814474   valid's QWK: 0.798249
[100]  train's QWK: 0.827018   valid's QWK: 0.802383
[125]  train's QWK: 0.837578   valid's QWK: 0.805626
[150]  train's QWK: 0.845204   valid's QWK: 0.807315
[175]  train's QWK: 0.853171   valid's QWK: 0.807868
[200]  train's QWK: 0.859088   valid's QWK: 0.808686
[225]  train's QWK: 0.864917   valid's QWK: 0.810234
[250]  train's QWK: 0.869947   valid's QWK: 0.81144
[275]  train's QWK: 0.87533    valid's QWK: 0.809521
[300]  train's QWK: 0.881035   valid's QWK: 0.809126
Early stopping, best iteration is:
[249]  train's QWK: 0.869933   valid's QWK: 0.811655
Evaluated only: QWK
```

Fold_3 Training =====

```
[LightGBM] [Info] Using self-defined objective function
Training until validation scores don't improve for 75 rounds
[25]   train's QWK: 0.756461   valid's QWK: 0.725361
[50]   train's QWK: 0.801151   valid's QWK: 0.759674
[75]   train's QWK: 0.81988    valid's QWK: 0.772748
[100]  train's QWK: 0.831812   valid's QWK: 0.777786
[125]  train's QWK: 0.840608   valid's QWK: 0.779492
[150]  train's QWK: 0.847348   valid's QWK: 0.785013
[175]  train's QWK: 0.854286   valid's QWK: 0.783577
[200]  train's QWK: 0.859245   valid's QWK: 0.785662
[225]  train's QWK: 0.864958   valid's QWK: 0.785597
[250]  train's QWK: 0.870441   valid's QWK: 0.786601
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[275]  train's QWK: 0.87637    valid's QWK: 0.784619
[300]  train's QWK: 0.881506   valid's QWK: 0.787956
[325]  train's QWK: 0.885256   valid's QWK: 0.788873
[350]  train's QWK: 0.889865   valid's QWK: 0.78892
[375]  train's QWK: 0.894752   valid's QWK: 0.789031
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[400]  train's QWK: 0.898468   valid's QWK: 0.788383
```

```
[425] train's QWK: 0.903077 valid's QWK: 0.78606
[450] train's QWK: 0.907371 valid's QWK: 0.78583
Early stopping, best iteration is:
[381] train's QWK: 0.895508 valid's QWK: 0.789993
Evaluated only: QWK
```

Fold_4 Training =====

```
[LightGBM] [Info] Using self-defined objective function
Training until validation scores don't improve for 75 rounds
[25] train's QWK: 0.754983 valid's QWK: 0.73259
[50] train's QWK: 0.799177 valid's QWK: 0.767048
[75] train's QWK: 0.818527 valid's QWK: 0.781897
[100] train's QWK: 0.83135 valid's QWK: 0.789811
[125] train's QWK: 0.840709 valid's QWK: 0.791026
[150] train's QWK: 0.846661 valid's QWK: 0.791864
[175] train's QWK: 0.853789 valid's QWK: 0.795351
[200] train's QWK: 0.860321 valid's QWK: 0.800228
[225] train's QWK: 0.865392 valid's QWK: 0.801369
[250] train's QWK: 0.871635 valid's QWK: 0.801595
[275] train's QWK: 0.876176 valid's QWK: 0.800188
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[300] train's QWK: 0.881295 valid's QWK: 0.799444
Early stopping, best iteration is:
[233] train's QWK: 0.867266 valid's QWK: 0.802263
Evaluated only: QWK
```

Fold_5 Training =====

```
[LightGBM] [Info] Using self-defined objective function
Training until validation scores don't improve for 75 rounds
[25] train's QWK: 0.755822 valid's QWK: 0.745336
[50] train's QWK: 0.799541 valid's QWK: 0.779397
[75] train's QWK: 0.818242 valid's QWK: 0.792132
[100] train's QWK: 0.827728 valid's QWK: 0.79735
[125] train's QWK: 0.838203 valid's QWK: 0.801416
[150] train's QWK: 0.846438 valid's QWK: 0.804186
[175] train's QWK: 0.852224 valid's QWK: 0.802931
[200] train's QWK: 0.858784 valid's QWK: 0.804392
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[225] train's QWK: 0.864979 valid's QWK: 0.806021
[250] train's QWK: 0.87072 valid's QWK: 0.805615
[275] train's QWK: 0.876262 valid's QWK: 0.80315
[300] train's QWK: 0.881617 valid's QWK: 0.80464
Early stopping, best iteration is:
[239] train's QWK: 0.868117 valid's QWK: 0.807126
Evaluated only: QWK
```

5.3 Validating LGBMRegressor model

```
In [ ]: preds, trues = [], []

for fold, model in enumerate(models):
    X_eval_cv = train_feats[train_feats["fold"] == fold].drop(columns=train_drop
    y_eval_cv = train_feats[train_feats["fold"] == fold]["score"]

    pred = model.predict(X_eval_cv) + a

    trues.extend(y_eval_cv)
```

```

    preds.extend(np.round(pred, 0))

v_score = cohen_kappa_score(trues, preds, weights="quadratic")

print(f"Validation score : {v_score}")

```

Validation score : 0.8026743411070119

5.4 Testing and collecting prediction

```
In [ ]: train_feats.shape
```

```
Out[ ]: (17307, 3907)
```

```
In [ ]: test_feats.shape
```

```
Out[ ]: (3, 3759)
```

```
In [ ]: X_eval_cv.shape
```

```
Out[ ]: (3461, 3754)
```

```

In [ ]: # Get the feature columns of the model
        model_columns = model.feature_name_

        # Get the feature columns of the input data
        input_columns = X_eval_cv.columns

        # Find extra features in the model
        extra_features_model = [col for col in model_columns if col not in input_columns]

        # Find extra features in the input data
        extra_features_input = [col for col in input_columns if col not in model_columns]

        # Print or inspect the extra features
        print("Extra features in the model:", extra_features_model)
        print("Extra features in the input data:", extra_features_input)

```

Extra features in the model: []

Extra features in the input data: []

```
In [ ]: len(extra_features_input)
```

```
Out[ ]: 0
```

```
In [ ]: len(extra_features_model)
```

```
Out[ ]: 0
```

```

In [ ]: # predicting for 5 models
        preds = []
        for fold, model in enumerate(models):
            X_eval_cv = test_feats.drop(columns=test_drop_columns)
            # pred = model.predict(X_eval_cv)
            pred = model.predict(X_eval_cv) + a
            preds.append(pred)

        # Combining the 5 model results

```

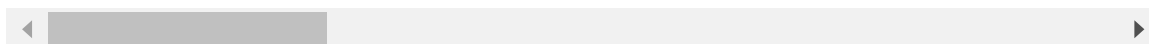
```
for i, pred in enumerate(preds):  
    test_feats[f"score_pred_{i}"] = pred  
test_feats["score"] = np.round(test_feats[[f"score_pred_{fold}" for fold in rang
```

In []: test_feats.head()

Out []:

	essay_id	paragraph_50_cnt	paragraph_75_cnt	paragraph_100_cnt	paragraph_125_cnt
0	000d118	1	1	1	1
1	000fe60	5	5	5	5
2	001ab80	4	4	4	4

3 rows × 3765 columns



6. Submission

In []: test_feats[["essay_id", "score"]].to_csv("submission.csv", index=False)

7. Save Model using Pickle

In []: `import joblib`
`joblib.dump(model, 'lgbm_model.pkl')`

Out []: ['lgbm_model.pkl']