AN INTELLIGENT ESSAY GRADING SYSTEM USING NLP AND SENTIMENT ANALYSIS (BACKEND IMPLEMENTATION)

Importing the necessary packages

In [1]:

```
#Numerical Python for array manipulations.
import numpy as np
#Pandas for reading the dataset and converting it into dataframes.
import pandas as pd
#Math library for rounding off to next whole number.
import math
#os module to specify the directory path.
import os

#Disables warnings
import warnings
warnings.filterwarnings("ignore")

#Natural Language Toolkit
import nltk
#stopwords - words that do not add much meaning to the sentence.
from nltk.corpus import stopwords
#PorterStemmer - removes common morphological endings from words (tense, number, pl
from nltk.stem import PorterStemmer
#SentimentIntensityAnalyzer - Implements and facilitates sentiment analysis tasks.
from nltk.sentiment.vader import SentimentIntensityAnalyzer
#nltk.download('stopwords')
#nltk.download('vader_lexicon')     #Model for text sentiment analysis
#nltk.download('punkt')             #Sentence tokenizer

#CountVectorizer - Tokenizes collection of text and builds vocabulary of known word
from sklearn.feature_extraction.text import CountVectorizer
#TfidfVectorizer - Highlight interesting words.
from sklearn.feature_extraction.text import TfidfVectorizer
#Provides train/text split for training.
from sklearn.model_selection import KFold
#Used to fit a linear model.
from sklearn.linear_model import LinearRegression
#cohen-kappa-score is used to measure agreement between two raters.
from sklearn.metrics import cohen_kappa_score

#Word2Vec creates word embeddings (Creates word vector for each word).
from gensim.models import Word2Vec
#KeyedVectors generates mapping between keys and vectors.
from gensim.models import KeyedVectors

#Embedding translates high dimensional vectors and makes it easy to do ML on large
from tensorflow.keras.layers import Embedding
#pad-sequences ensures all sequences in a list have same length.
from tensorflow.keras.preprocessing.sequence import pad_sequences
#A sequential model is used.
from tensorflow.keras.models import Sequential
#one-hot is used to one hot encode categorical values.
from tensorflow.keras.preprocessing.text import one_hot
#LSTM layers for building the model.
#Dropout layer to prevent overfitting.
#Dense layers as the output layer.
from tensorflow.keras.layers import LSTM, Dropout, Dense

#Lambda (Creates a nameless fuction for a short period of time).
#Flatten layer is used to compress the input into 1D vector.
from keras.layers import Lambda, Flatten
#load_model is used to load a saved model.
#model_from_config instantiates a keras model from its config.
```

```
from keras.models import load_model, model_from_config
#Keras backend API.
import keras.backend as K

#Regular Expressions.
import re

#python spell checker.
#from spellchecker import SpellChecker

#Python grammar checker.
import language_tool_python

#Allows to send HTTP requests.
import requests
#Helps to fetch data from XML and HTML files.
from bs4 import BeautifulSoup as bs

#Used to compare a pair of inputs.
from difflib import SequenceMatcher
```

```
Using TensorFlow backend.
```

Reading the dataset

In [2]:

```
df = pd.read_csv("./fyp/training_set_rel3.tsv", sep='\t', encoding='ISO-8859-1')
```

Removing the tagged labels and word tokenizing the sentence

In [3]:

```
def essay_wordlist(essay_1, rem_stopwords):
    #match all strings without a letter and replace it with a white space character
    essay_1 = re.sub("[^A-Za-z]", " ", essay_1)
    #Convert the essay into all lower case characters.
    words = essay_1.lower().split()
    #Removing stop words from the essay.
    if rem_stopwords:
        #creates a set of stopwords in english.
        stop = set(stopwords.words("english"))
        #reassigns an essay containing no stop words.
        words = [word1 for word1 in words if not word1 in stop]

    #return the words list.
    return (words)
```

Sentence tokenizing the essay and word tokenization

In [4]:

```
def essay_sentences(essay_1, rem_stopwords):
    #Load the pre-trained punkt tokenizer for English.
    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    #Tokenizing the essay.
    sentence = tokenizer.tokenize(essay_1.strip())
    sentences = []
    #Generate word list for the tokenizer sentences.
    for sentence1 in sentence:
        if len(sentence1) > 0:
            sentences.append(essay_wordlist(sentence1, rem_stopwords))
    #Return the sentence list.
    return sentences
```

Making feature vector from the words list of an essay

In [5]:

```
def FeatureVector(words, model, no_feat):
    #Create an array filled with zeroes.
    FeatureVector = np.zeros((no_feat,),dtype="float32")
    no_words = 0.
    #Convert the list of names in the vocabulary into a set.
    indextoword_set = set(model.wv.index2word)
    #Calculate the word count and if a word is present in the vocabulary add it to
    for x in words:
        if x in indextoword_set:
            no_words  = no_words + 1
            FeatureVector = np.add(FeatureVector,model[x])
    #Calculate the average.
    FeatureVector = np.divide(FeatureVector,no_words)
    #Return the feature vector.
    return FeatureVector
```

Main function to generate the word vectors for word2vec model

In [6]:

```
def AvgFeatureVectors(essays, model, no_feat):
    flag = 0
    #Create another array with dimensions length of essay and number of features fi
    FeatureVectors = np.zeros((len(essays),no_feat),dtype="float32")
    #For each essay append the average feature vector into the FeatureVector array.
    for x in essays:
        FeatureVectors[flag] = FeatureVector(x, model, no_feat)
        flag = flag + 1
    #Return the total average feature vector.
    return FeatureVectors
```

Defining the model

In [7]:

```
def define_model():
    #Declare a sequential model.
    model = Sequential()
    #Add two LSTM layers a dropout layer and a dense layer with rectified linear un
    model.add(LSTM(300, dropout=0.4, recurrent_dropout=0.4, input_shape=[1, 300], r
    model.add(LSTM(64, recurrent_dropout=0.3))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='relu'))

    #Compile the model
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['accurac
    model.summary()

    #Return the defined model.
    return model
X=df
y = df['domain1_score']
```

Training the model

In [8]:

```
#Define 5 splits for KFOLD training.
x = KFold(n_splits = 5, shuffle = True)
output = []
y_pred1 = []

fold = 1
#Perform training by creating a list from the dataset for each train and test datas
for train, test in x.split(X):
    print("\nFold {}\n".format(fold))
    #Declare test and train sets for each fold.
    x_train, x_test, y_train, y_test = X.iloc[train], X.iloc[test], y.iloc[train],

    #Define the test and train essays from the 'essay' column of the dataset.
    training_essays = x_train['essay']
    testing_essays = x_test['essay']

    a = []

    #Sentence tokenize each training essay.
    for essay in training_essays:
            a = a + essay_sentences(essay, rem_stopwords = True)

    no_feat = 300
    word_count = 40
    no_workers = 4
    cont = 10
    sample = 1e-3

    #Predict the nearby words for each word in the sentence.
    model = Word2Vec(a, workers=no_workers, size=no_feat, min_count = word_count, w

    #Normalize vectors (Equal length)
    model.init_sims(replace=True)
    #Save the model.
    model.wv.save_word2vec_format('word2vecmodel.bin', binary=True)

    cleaning_train_essays = []

    #For each training essay generate a word list.
    for essay_1 in training_essays:
        cleaning_train_essays.append(essay_wordlist(essay_1, rem_stopwords=True))
    #Generate average feature vectors for the word lists.
    Vectors_train = AvgFeatureVectors(cleaning_train_essays, model, no_feat)

    #Similarly for the test essays generate word lists and average feature vectors.
    cleaning_test_essays = []
    for essay_1 in testing_essays:
        cleaning_test_essays.append(essay_wordlist( essay_1, rem_stopwords=True ))
    Vectors_test = AvgFeatureVectors( cleaning_test_essays, model, no_feat )

    #Reshape the average feature vectors of test and train datasets to the shape of
    Vectors_train = np.array(Vectors_train)
    Vectors_test = np.array(Vectors_test)
    Vectors_train = np.reshape(Vectors_train, (Vectors_train.shape[0], 1, Vectors_t
    Vectors_test = np.reshape(Vectors_test, (Vectors_test.shape[0], 1, Vectors_test

    #Assign the defined model.
    lstm_model = define_model()
    #Fit the model.
```

```
    lstm_model.fit(Vectors_train, y_train, batch_size=64, epochs=10)
    #Load the model weights.
    #lstm_model.load_weights('./fyp/model.h5')
    y_predict = lstm_model.predict(Vectors_test)

    #Save the model when all the folds are completed.
    if fold == 5:
         lstm_model.save('./fyp/model.h5')

    #Round off the predicted value.
    y_predict = np.around(y_predict)

    #Generate a kappa score for each fold.
    result = cohen_kappa_score(y_test.values,y_predict,weights='quadratic')
    print("Kappa Score for fold {fold} is {score}".format(fold = fold, score = resu
    #Add each kappa score to the overall score.
    output.append(result)

    #Increment the value of fold.
    fold = fold + 1
```

```
Fold 1


_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 300)            721200
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                93440
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 32)                2080
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 816,753
Trainable params: 816,753
Non-trainable params: 0
```

Average kappa score

In [9]:

```
print("Average Kappa score : ",np.around(np.array(output).mean(),decimals=3))
```

```
Average Kappa score :  0.905
```

Initializing an essay for grading

In [23]:

```
#Essay for grading.

#content = "Dear@CAPS1 @CAPS2, I believe that using computers will benefit us in ma
content = "When I realized I cannot understand the world. I recently debated at the
#content = "Dear @CAPS1, Computers can have a positive effect or a negative effect
#content = "I think computers are good because you can talk to your friends and fam
#content = "When I realized I was a punk rocker philosopher. One summer night, my f
content
```

Out[23]:

```
"When I realized I cannot understand the world. I recently debated at
the Orange County Speech League Tournament, within the Parliamentary D
ivision. This specific branch of debate is an hour long, and consists
of two parties debating either side of a current political issue. In o
ne particular debate, I was assigned the topic: “Should Nation States
eliminate nuclear arms?” It so happened that I was on the negative sid
e and it was my job to convince the judges that countries should conti
nue manufacturing nuclear weapons. During the debate, something strang
e happened: I realized that we are a special breed of species, that so
much effort and resources are invested to ensure mutual destruction. A
nd I felt that this debate in a small college classroom had elucidated
something much more profound about the scale of human existence. In an
y case, I won 1st place at the tournament, but as the crowd cheered wh
en my name was called to stand before an audience of hundreds of other
debaters, and I flashed a victorious smile at the cameras, I couldn’t
help but imagine that somewhere at that moment a nuclear bomb was bein
g manufactured, adding to an ever-growing stockpile of doom. And tha
t's when I realized that the world was something I will never understa
nd."
```

Generating grade prediction

In [24]:

```
#Load the saved word2vec model.
model = KeyedVectors.load_word2vec_format( "./word2vecmodel.bin", binary=True)
test_essays = []
#Create a word list, average input features and reshape the input essay.
test_essays.append(essay_wordlist( content, rem_stopwords=True ))
test_vectors = AvgFeatureVectors( test_essays, model, no_feat )
test_vectors = np.array(test_vectors)
test_vectors = np.reshape(test_vectors, (test_vectors.shape[0], 1, test_vectors.sha

#Load the weights of the LSTM model.
lstm_model.load_weights("./fyp/model.h5")
#Generate grade prediction for the input essay.
preds = lstm_model.predict(test_vectors)
```

In [ ]:

```

```

Detecting Spelling and Grammatical errors

In [25]:

```
tool = language_tool_python.LanguageTool('en-US')

#Parsing the input essay to detect syntactic errors.
matches = tool.check(content)

#Displaying the errors found.
for match in matches:
    print(match)
    print("\n")
```

```
Offset 0, length 4, Rule ID: SENTENCE_FRAGMENT
Message: “When” at the beginning of a sentence usually requires a 2nd
clause. Maybe a comma, question or exclamation mark is missing, or the
sentence is incomplete and should be joined with the following sentenc
e.
When I realized I cannot understand the worl...
^^^^


Offset 416, length 8, Rule ID: COMMA_COMPOUND_SENTENCE
Message: Use a comma before ‘and’ if it connects two independent claus
es (unless they are closely connected and short).
Suggestion: side, and
... so happened that I was on the negative side and it was my job to c
onvince the judges th...
                                            ^^^^^^^^
```

Sentiment analysis

In [26]:

```
#Initialize the Sentiment Analyzer
sid = SentimentIntensityAnalyzer()
#Generate polarities for the input essay.
polarities = sid.polarity_scores(content)
```

Printing the grade

In [27]:

```
print("Grade   Scored\n")
#Generate grade prediction.
x = math.ceil(preds/2)
y = ""
#If the predicted score is less than 5 the grade is poor.
if x < 5:
    y = "Poor"
#Else if the predicted score is between 5 and 8 the grade is average.
elif x >= 5 and x < 8:
    y = "Average"
#If the predicted score is greater than or equal to 8 then the grade is Excellent.
else:
    y = "Excellent"

print(x, "\t", y, "\n")
#Display the positive, Negative, Neutral and compound polarities.
for key in sorted(polarities):
        print('{0}: {1}, '.format(key, polarities[key]), end='')
#If the compound polarity is greater than or equal to 0.6, the approach is positive
if polarities['compound'] >= 0.6:
            print("\n\nYour approach towards the topic is considered positive and a
#If the compound polarity is less than or equal to 0.4, the approach is negative.
elif polarities['compound'] <= 0.4:
            print("\n\nYour approach towards the topic is considered negative")
#If the compound polarity id near 0.5, the appriach is neutral.
else:
            print("\n\nYour approach towards the topic is considered neutral and re
```

```
Grade   Scored

6        Average

compound: 0.5719, neg: 0.073, neu: 0.832, pos: 0.095,

Your approach towards the topic is considered neutral and relevant
```

In [ ]:

```

```

Corrected essay

In [28]:

```
mistakes = []
corrections = []
positions1 = []
positions2 = []

#For each syntactical mistake in the essay, replace the mistake with the appropriat
for a in matches:
    if len(a.replacements)>0:
        positions1.append(a.offset)
        positions2.append(a.errorLength+a.offset)
        mistakes.append(content[a.offset:a.errorLength+a.offset])
        corrections.append(a.replacements[0])


#Create a list of the input essay.
new_text = list(content)

#Create a new string of text based on the values in the mistakes list and the origi
for m in range(len(positions1)):
    for i in range(len(content)):
        new_text[positions1[m]] = corrections[m]
        if (i>positions1[m] and i<positions2[m]):
            new_text[i]=""

new_text = "".join(new_text)
new_text
```

Out[28]:

```
"When I realized I cannot understand the world. I recently debated at
the Orange County Speech League Tournament, within the Parliamentary D
ivision. This specific branch of debate is an hour long, and consists
of two parties debating either side of a current political issue. In o
ne particular debate, I was assigned the topic: “Should Nation States
eliminate nuclear arms?” It so happened that I was on the negative sid
e, and it was my job to convince the judges that countries should cont
inue manufacturing nuclear weapons. During the debate, something stran
ge happened: I realized that we are a special breed of species, that s
o much effort and resources are invested to ensure mutual destruction.
And I felt that this debate in a small college classroom had elucidate
d something much more profound about the scale of human existence. In
any case, I won 1st place at the tournament, but as the crowd cheered
when my name was called to stand before an audience of hundreds of oth
er debaters, and I flashed a victorious smile at the cameras, I could
n’t help but imagine that somewhere at that moment a nuclear bomb was
being manufactured, adding to an ever-growing stockpile of doom. And t
hat's when I realized that the world was something I will never unders
tand."
```

Plagiarism Detection

Searching for websites with similar content

In [29]:

```
def search(query, num):

    #Define a URL to perform searching
    url = 'https://www.bing.com/search?q=' + query
    url1 = []

    #Generae a HTTP request to the URL
    x = requests.get(url, headers = {'User-agent': 'John Doe'})
    #Fetch the data in the site using beautifulsoup.
    y = bs(x.text, 'html.parser')

    #Append the current examined URL into a list.
    for a in y.find_all('a'):
        url = str(a.get('href'))
        if url.startswith('http'):
            if not url.startswith('http://go.m') and not url.startswith('https://go
                url1.append(url)

    return url1[:num]
```

Extracting text from the detected websites

In [30]:

```
def extract(url):
    x = requests.get(url)
    y = bs(x.text, 'html.parser')
    #Return the text from the site.
    return y.get_text()
#Define Stopping words.
stopping_words = set(nltk.corpus.stopwords.words('english'))
```

tokenizing the text for comparison

In [31]:

```
def TokenizeText(string):
    #Tokenize the string in the site.
    words = nltk.word_tokenize(string)
    #Return all the non stopping words.
    return (" ".join([word for word in words if word not in stopping_words]))
```

Comparing the text

In [32]:

```
def Verify(string, results_per_sentence):
    #Sentence tokenize the string in the site.
    sentences = nltk.sent_tokenize(string)
    matching_sites = []
    #Detect URLs where similar content is found.
    for url in search(query=string, num=results_per_sentence):
        matching_sites.append(url)
    #Detct the sentences in the URL.
    for sentence in sentences:
        for url in search(query = sentence, num = results_per_sentence):
            matching_sites.append(url)

    #Return the URLs
    return (list(set(matching_sites)))

def similarity(str1, str2):
    #Match the entire (100%) two contents and return the ratio of similarities betw
    return (SequenceMatcher(None,str1,str2).ratio())*100
```

Generating the report of plagiarized content

In [33]:

```
def result(text):

    #Copare the two texts.
    matching_sites = Verify(TokenizeText(text), 2)
    matches = {}

    #For each matching site determine the amount of similarity.
    for i in range(len(matching_sites)):
        matches[matching_sites[i]] = similarity(text, extract(matching_sites[i]))

    #Sort the similarities in descending order
    matches = {k: v for k, v in sorted(matches.items(), key=lambda item: item[1], r

    #Return the URLs and their corresponding plagiarized percentage score as a dict
    return matches
```

Printing the plagiarism result

In [34]:

```
plag = {}
total = 0
plag = result(content)
#For each key, value pair in the dictionary, display the URL and the amount of % pl
for key, value in plag.items():
    print(key, value, "%")
    #Calculate the total plagiarism percentage.
    total = total+value
#Display the total Plagiarism percentage.
print("Total Plagiarism Detected ", total, "%")
```

```
https://collegeessayguy.tumblr.com/post/101139595504/personal-statemen
t-example-punk-rock (https://collegeessayguy.tumblr.com/post/101139595
504/personal-statement-example-punk-rock) 27.730169193583826 %
https://www.currentschoolnews.com/education-news/college-entrance-essa
y-writing-tips/ (https://www.currentschoolnews.com/education-news/coll
ege-entrance-essay-writing-tips/) 9.496158656423084 %
https://www.coursehero.com/file/81746153/1docx/ (https://www.courseher
o.com/file/81746153/1docx/) 4.017857142857143 %
https://www.coursehero.com/file/82775212/course-hero-8docx/ (https://w
ww.coursehero.com/file/82775212/course-hero-8docx/) 4.017857142857143
 %
https://www.collegeessayguy.com/blog/college-essay-examples (https://w
ww.collegeessayguy.com/blog/college-essay-examples) 2.272522806196808
 %
https://www.religiousforums.com/threads/i-just-realized-i-dont-underst
and-this-world.238156/ (https://www.religiousforums.com/threads/i-just
-realized-i-dont-understand-this-world.238156/) 0.9662576687116564 %
https://tinybuddha.com/blog/the-best-thing-to-say-to-someone-who-wont-
understand-you/ (https://tinybuddha.com/blog/the-best-thing-to-say-to-
someone-who-wont-understand-you/) 0.8666666666666666 %
https://www.elitedaily.com/life/motivation/cant-understand-experience-
yourself/954658 (https://www.elitedaily.com/life/motivation/cant-under
stand-experience-yourself/954658) 0.5161290322580645 %
https://psiloveyou.xyz/10-life-quotes-that-if-applied-will-change-the-
way-you-see-the-world-forever-d05338ae489b (https://psiloveyou.xyz/10-
life-quotes-that-if-applied-will-change-the-way-you-see-the-world-fore
ver-d05338ae489b) 0.45062806286261475 %
Total Plagiarism Detected  50.33424637241701 %
```

In [ ]: