ADVANCEMENTS IN PLAGIARISM DETECTION USING MACHINE LEARNING

BY

BURRA ABHISHEK - 19BCE1187

SIDDHANT ROY - 19BCE1181

A project report submitted to

Dr. BHARADWAJA KUMAR

SCHOOL OF ELECTRONICS AND COMPUTER ENGINEERING

In partial fulfillment of the requirements for the course of CSE4022 –

Natural Language Processing

VIT UNIVERSITY, CHENNAI CAMPUS

Vandalur-Kelambakkam Road

Chennai-600127

December 2021

BONAFIDE CERTIFICATE

Certified that this project report entitled "Advancements in Plagiarism Detection using Machine Learning" is a bonafide work of **Burra Abhishek (19BCE1187) and Siddhant Roy (19BCE1181)** who carried out the J-component under my supervision and guidance.

Dr. BHARADWAJA KUMAR

Professor

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCSE)

VIT UNIVERSITY, CHENNAI

CAMPUSCHENNAI-600127

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Bharadwaja Kumar**, Professor, SCSE, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to the Dean of the SCOPE, VIT Chennai, for extending the facilities of the School towards our project and for the unstinting support. We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

ABSTRACT

In the age of digital technologies, plagiarism detection ensures originality and creativity in document submissions. While current algorithms can detect lexical plagiarism, efficient algorithms to detect syntactic plagiarism are missing. Furthermore, current algorithms treat citations and references to research papers and journals as plagiarized text. Online tools have word limits up to two thousand words and most of them serve ads or hide features behind paywalls. This paper presents a classification-based method to detect syntactic plagiarism and avoid false positives. It also demonstrates a forever free, adless and open-source online tool to detect plagiarism using this method.

INDEX

KEYWORDS

Natural Language Processing: The branch of Artificial Intelligence which helps computers process and understand text and spoken words similar to how humans can.

Machine Learning: The branch of Artificial Intelligence that deals with the learning of computer algorithms based on its experience and data usage.

Plagiarism Detection: Plagiarism is the representation of another author’s idea or work as one’s own work. Plagiarism detection deals with the detection of plagiarized text from a given text or paragraph.

Paraphrase Detection: The detection of paraphrases – sentences with different words but the same meaning.

Free software: Software that guarantees users freedom, not just in price, but also in the freedom to share, copy and use the work as the user wants.

1. INTRODUCTION

In today’s world, everything is moving online from digital transactions to examinations and research. With the onset of lockdowns to prevent the spread of the coronaviruses, almost all activities were shifted online. This means that students have easy access to online resources, which they unfortunately misuse and plagiarize. Turnitin, a very popular plagiarism checker amongst academia, is very costly [1]. It’s award-winning plagiarism checker iThenticate starts at $100 per document [2]. Other services may be free, but the free versions have restrictions. For example, the plagiarism checker provided by https://www.check-plagiarism.com/ is much faster and provides more search terms if one pays for their services, and it shows advertisements [3]. The plagiarism checker provided by https://smallseotools.com/ imposes a word limit of 1,000 for free checking and increases the limit varying from 25,000 to 510,000 words depending on the amount paid for the service, and it shows advertisements [4]. During the pandemic, several families are not able to earn a decent living. Therefore, I propose to build a free and open-source plagiarism detection based on paraphrase detection.

2.DATA SETS

Two datasets were used. One is to train the machine learning models, and the other is to generate vectors for training a gensim model for computing the Normalized Word Mover's Distance.

For training the machine learning models, we used the PAWS QQP (Paraphrase Adversaries from Word Scrambling – Quora Question Pairs corpus) dataset from the Google Research Datasets repository [8][9]. The training data has 49401 entries. Each entry consists of two sentences (indices 0 and 1) and the paraphrase coefficient (noisy label) (index: 2, 0 – not paraphrase, 1 – paraphrase).

For the Normalized Word Mover's Distance, we use the GloVe dataset (Global Vectors for word representation) from Stanford NLP repository [10]. Due to speed being a factor in the paraphrase detection, we use the smaller GloVe6B dataset (Wikipedia 2014 + Gigaword 5). Since this model contains vectors mapped to text, the text format should first be converted to the vector format which is easily understood by gensim before training the vectors for the Word Mover's Distance

3. METHODOLOGY

Given a text to check, the following procedure is implemented:

1. Text pre-processing

   The pre-processing of text includes replacing “\r” and “\n” with spaces and splitting the text into sentences based on sentence boundaries, removal of punctuation marks, bullet points, and unnecessary whitespaces, and the conversion of accented characters to ASCII values. This gives the best performance [7].
   In addition to the usual pre-processing, topics like references, citations, conflicts of interest, and other similar topics with a common response are ignored by the proposed model.

2. Web mining

   We search online for different webpages using the googlesearch library provided by Python. This searches for the URLs using Google. The first URL is chosen (the remaining are not considered for performance reasons) and the mime type is determined from the headers using BeautifulSoup library. Most of them are in PDF format, while some may be in HTML or XML format. The text must be extracted from each of these formats. Parsing PDF text requires a PDF parser which works accurately on complex PDF files.

3. Model training

   We used the PAWS QQP dataset in [8][9] to train the models. This corpus dataset consists of 49401 items in the training part. Each of these items consists of the first sentence (index: 0), the second sentence (index: 1) and whether they both are paraphrases or not (index: 2).
   Each sentence is pre-processed similar to the text pre-processing.

4. Lexical analysis

   Various lexical features are used, including fuzzy generation, n-gram models and normalized longest common subsequence.
   For fuzzy generation, fuzz ratio and fuzz token set ratio are used. The fuzz ratio is determined by the similarity of the entire strings being compared.

Fuzz token set ratio determines the similarity of each token involved without considering word order. All these fuzz ratios use Levenshtein's algorithm to compute the string similarity – for one step, we may add, remove or edit a character in the string, and we want to determine the number of steps it takes to transform the string from one step to another. For the n-gram models, we generate unigrams, bigrams and trigrams out of these sentences, and we determine the common n-gram ratio and the Jaccard distance in each case. Jaccard distance between two n-grams is computed as the quotient of their set intersection divided by their set union. The longest common subsequence is similar to the longest common substring, but the character position in this case need not be consecutive.

5. Semantic analysis

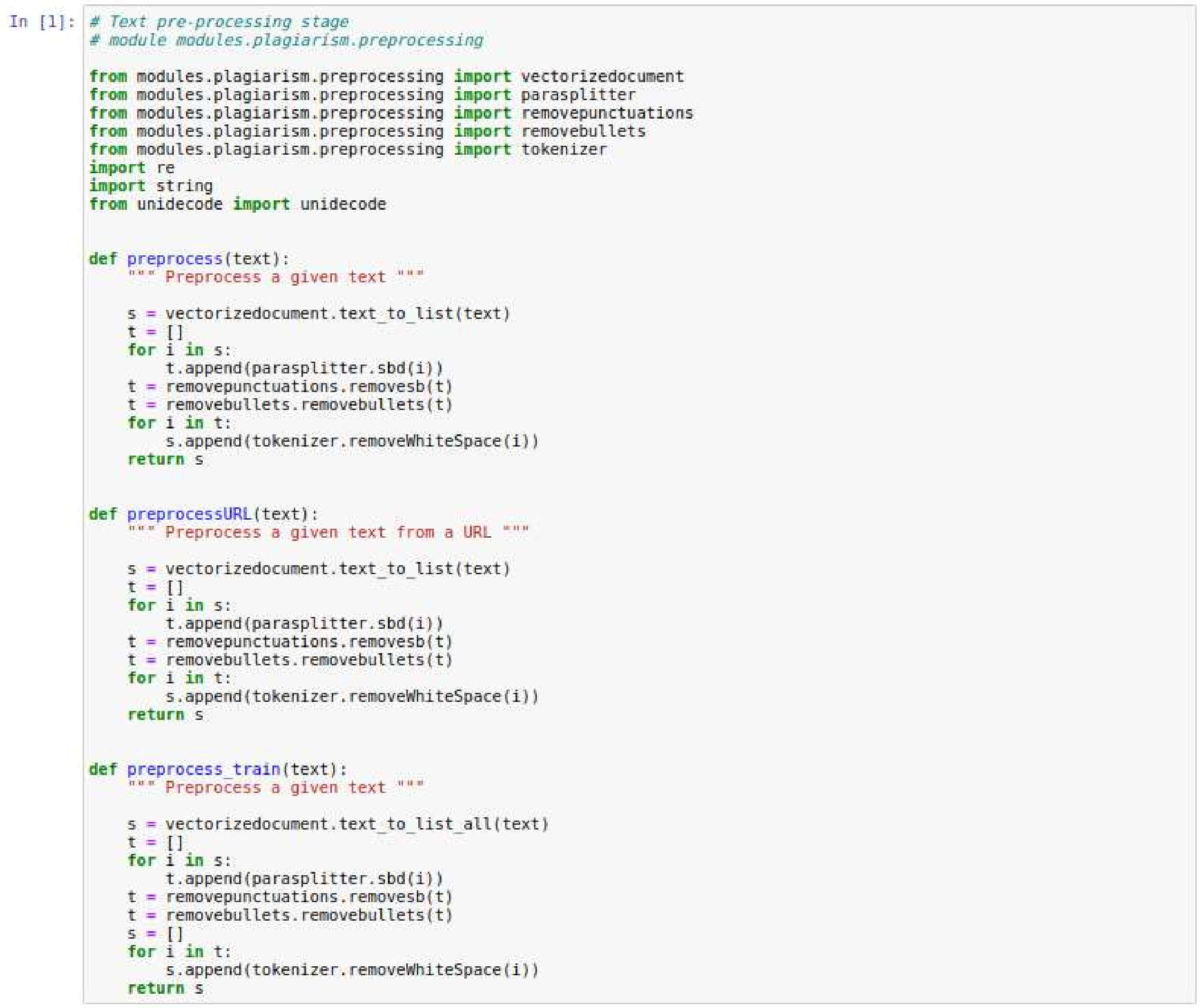
   From the research in [7], Normalized Word Mover Distance has maximum contribution to the paraphrase detection over all other semantic analyses. For computing the Normalized Word Mover Distance, we use the GloVe6b dataset. Due to size and time constraints, we use the 50-dimension dataset (171 MB) [10]. The text file is first converted into word2vec (vector format) using the one-line code in [11], and then loaded using gensim. The Normalized Word Mover Distance between two sentences are computed using this model by measuring their semantic distance using the model embeddings.
   The other features and syntactic analysis were skipped for performance reasons.

6. Plagiarism reporting

   Finally, we use Logistic Regression and Random Forest Classifier to compute the paraphrase similarity between the given sentence and each of the sentences obtained through web mining. As a final check, we ensure that sentences which are exact copies of each other are flagged as paraphrases. A score of 1 would mean paraphrase, while a score of 0 would mean not paraphrase. Should any of the online references indicate a paraphrase, the sentence is flagged as plagiarized. The percentage of plagiarized sentences out of the entire document is returned as a plagiarism report.

4. CODE

The entire source code is available on GitHub at https://github.com/burraabhishek/lipyplagiarism. For this report, the branch "b2_projectsubmission" is used. Screenshots of the Jupyter Notebook are attached here:



```
In [1]: # Text pre-processing stage
        # module modules.plagiarism.preprocessing

        from modules.plagiarism.preprocessing import vectorizedocument
        from modules.plagiarism.preprocessing import parasplitter
        from modules.plagiarism.preprocessing import removepunctuations
        from modules.plagiarism.preprocessing import removebullets
        from modules.plagiarism.preprocessing import tokenizer
        import re
        import string
        from unidecode import unidecode


        def preprocess(text):
            """ Preprocess a given text """

            s = vectorizedocument.text_to_list(text)
            t = []
            for i in s:
                t.append(parasplitter.sbd(i))
            t = removepunctuations.removesb(t)
            t = removebullets.removebullets(t)
            for i in t:
                s.append(tokenizer.removeWhiteSpace(i))
            return s


        def preprocessURL(text):
            """ Preprocess a given text from a URL """

            s = vectorizedocument.text_to_list(text)
            t = []
            for i in s:
                t.append(parasplitter.sbd(i))
            t = removepunctuations.removesb(t)
            t = removebullets.removebullets(t)
            for i in t:
                s.append(tokenizer.removeWhiteSpace(i))
            return s


        def preprocess_train(text):
            """ Preprocess a given text """

            s = vectorizedocument.text_to_list_all(text)
            t = []
            for i in s:
                t.append(parasplitter.sbd(i))
            t = removepunctuations.removesb(t)
            t = removebullets.removebullets(t)
            s = []
            for i in t:
                s.append(tokenizer.removeWhiteSpace(i))
            return s
```

In [2]:
```python
# Training the model

import pandas as pd
from modules.plagiarism.preprocessing import preprocess
from modules.plagiarism.comparisons import ds_splitter
from modules.plagiarism.comparisons.classifiers import logistic, randomforest
from sklearn import metrics


def get_train_df(location):
    train_file = open(location, mode='r')
    unprocessed_train_data = list([example.split("\t") for example in train_file.readlines()])[1:]
    for i in unprocessed_train_data:
        i.pop(0)
        i[2] = int(i[2].replace("\n", ""))
    return unprocessed_train_data


def process_dataset(dataset):
    for i in dataset:
        i[0] = preprocess.preprocess_train(i[0])[0]
        i[1] = preprocess.preprocess_train(i[1])[0]
    return dataset

# PAWS QQP dataset was used to train the models.
y = process_dataset(get_train_df('modules/plagiarism/comparisons/datasets/final/train.tsv'))
y1 = process_dataset(get_train_df('modules/plagiarism/comparisons/datasets/final/test.tsv'))
```

In [3]:
```python
pd.DataFrame(y)
```

Out[3]:

	0	1	2
0	in paris in october 1560 he secretly met the e...	in october 1560 he secretly met with the engli...	0
1	the nba season of 1975 76 was the 30th season ...	the 1975 76 season of the national basketball ...	1
2	there are also specific discussions public pro...	there are also public discussions profile spec...	0
3	when comparable rates of flow can be maintaine...	the results are high when comparable flow rate...	1
4	it is the seat of zerendi district in akmola r...	it is the seat of the district of zerendi in a...	1
...	...	...	...
49396	our school is of spiritual and spiritual love ...	our school is of the temporal and the spiritua...	0
49397	she was in cork on june 24 and arrived on 8 ju...	she was at cork on 24 june and arrived in the ...	1
49398	cornelia stuyvesant vanderbilt george and edit...	john john f a cecil the only child of george a...	0
49399	the third season was premiered on 7 june 2010 ...	the fourth season was premiered on june 7 2010	0
49400	it is also from a location on the mainland los...	it is also known from one location on the main...	0

49401 rows × 3 columns

In [4]:
```python
pd.DataFrame(y1)
```

Out[4]:

	0	1	2
0	this was a series of nested angular standards ...	this was a series of nested polar scales so th...	0
1	his father emigrated to missouri in 1868 but r...	his father emigrated to america in 1868 but re...	0
2	in january 2011 the deputy secretary general o...	in january 2011 fiba asia deputy secretary gen...	1
3	steiner argued that in the right circumstances...	steiner held that the spiritual world can be r...	0
4	luciano williames dias born july 25 1970 is a ...	luciano williames dias born 25 july 1970 is a ...	0
...	...	...	...
7995	the company has branches in tokyo based in the...	the company has branches in tokyo based in sai...	1
7996	muara teweh abbreviated mtw is a city located ...	teweh abbreviated mtw is a city located in the...	0
7997	the modern coat of arms of bavaria was designe...	the modern coat of arms of bavaria was designe...	1
7998	former president brenda kuecks received a clea...	in 2013 former president brenda kuecks receive...	0
7999	it is located near point pleasant borough a mu...	it is near point pleasant borough a municipali...	1

8000 rows × 3 columns

In [5]:
```python
# Lexical analysis

# Fuzzy similarity

from fuzzywuzzy import fuzz
from fuzzywuzzy import process


def fuzzysimilarity(dataframe):
    """
    Compares two strings in a Pandas DataFrame: dataframe[0], dataframe[1]
    String comparison using Levenshtein distance
    to calculate distance between sequences.
    We need all ratios
    """

    # Fuzz ratio: similarity of entire string
    dataframe['Fuzz Ratio'] = dataframe.apply(lambda x: fuzz.ratio(str(x[0]), str(x[1])), axis=1)

    # Fuzz Token Set Ratio: similarity of each token in the string
    # The word order does not matter, unlike in fuzz ratio
    dataframe['Fuzz Token Set Ratio'] = dataframe.apply(
        lambda x: fuzz.token_set_ratio(str(x[0]), str(x[1])),
        axis=1
        )

    return dataframe


# N-gram features and Jaccard similarity

from nltk.util import ngrams

def jaccardDistance(x, y, n):
    _w1, _w2, a, b = common_ngrams(x, y, n)
    l = len(set(a).union(set(b)))
    if l == 0:
        return 1
    else:
        return len(set(a).intersection(set(b))) / l


def ngrams_ratio(a, b, n):
    w1, w2, ngrams1, ngrams2 = common_ngrams(a, b, n)
    return len(set(ngrams1).intersection(set(ngrams2))) / (len(w1) + len(w2))


def common_ngrams(a, b, n):
    # Split the sentences into words
    w1 = a.split()
    w2 = b.split()
    # Get the n grams
    ngrams1 = list(ngrams(w1, n))
    ngrams2 = list(ngrams(w2, n))
    return w1, w2, ngrams1, ngrams2
```

```python
def ngram_features(df):
    df['Common Unigram Ratio'] = df.apply(
        lambda x: ngrams_ratio(str(x[0]), str(x[1]), 1),
        axis=1
        )
    df['Common Bigram Ratio'] = df.apply(
        lambda x: ngrams_ratio(str(x[0]), str(x[1]), 2),
        axis=1
        )
    df['Common Trigram Ratio'] = df.apply(
        lambda x: ngrams_ratio(str(x[0]), str(x[1]), 3),
        axis=1
        )
    df['Unigram Jaccard Distance'] = df.apply(
        lambda x: jaccardDistance(str(x[0]), str(x[1]), 1),
        axis=1
        )
    df['Bigram Jaccard Distance'] = df.apply(
        lambda x: jaccardDistance(str(x[0]), str(x[1]), 2),
        axis=1
        )
    df['Trigram Jaccard Distance'] = df.apply(
        lambda x: jaccardDistance(str(x[0]), str(x[1]), 3),
        axis=1
        )
    return df
```

```python
# Normalized Longest Common Subsequence

def NLCS(sentence1, sentence2):
    """ Determine the length of the NLCS of two sentences """

    # Get each individual word from each of the sentences.
    word1 = sentence1.split()
    word2 = sentence2.split()

    # Get the number of words from each sentence.
    l1 = len(word1)
    l2 = len(word2)

    # Initialize the nested list to store all the
    # subsequence similarity values
    a = []
    for i in range(l1 + 1):
        l = []
        for j in range(l2 + 1):
            l.append([])
        a.append(l)

    for i in range(l1 + 1):
        for j in range(l2 + 1):
            # Nothing to compare initially
            if i == 0 or j == 0:
                a[i][j] = 0
            # Matching words
            # Add 1 to the subsequence
            elif word1[i - 1] == word2[j - 1]:
                a[i][j] = a[i - 1][j - 1] + 1
            # Words do not match
            # Get the maximum value of its previous neighbours
            else:
                a[i][j] = max(a[i-1][j], a[i][j-1])

    # a[l1][l2] contains the length of the
    # longest common subsequence of X[0..n-1] & Y[0..m-1]
    lf = a[l1][l2]/(len((set(word1).union(set(word2)))))

    # lf is the length of the Normalized longest common subsequence
    return lf


def apply_nlcs(dataframe):
    dataframe["Normalized Longest Common Subsequence"] = dataframe.apply(lambda x: NLCS(x[0], x[1]), axis=1)
    return dataframe
```

In [6]:
```python
# Semantic Analysis

# Normalized Word Mover's Distance

from gensim import models

# If this code fails, you need to download the GloVe dataset.
# Open a file explorer, then go to modules/plagiarism/comparisons/datasets/
# Create a new folder "glove6b" and go to the following link to download the dataset:
# https://nlp.stanford.edu/data/wordvecs/glove.6B.zip
# Then use the 50d model. If any other model is used, change the file name accordingly.
# Before you can use the model, open a terminal or its equivalent in the same folder, then run
# python -m gensim.scripts.glove2word2vec --input  glove.6B.50d.txt --output glove.6B.50d.w2vformat.txt
w = models.KeyedVectors.load_word2vec_format(
    'modules/plagiarism/comparisons/datasets/glove6b/glove.6B.50d.w2vformat.txt', binary=False)

# w.init_sims may throw a deprecation warning, use fill_norms instead.
#w.init_sims(replace=True)
w.fill_norms()

def nwmd(dataframe):
    dataframe["Normalized Word Mover's Distance"] = dataframe.apply(
        lambda x: w.wmdistance(x[0], x[1]),
        axis=1
        )
    return dataframe
```

In [7]:
```
# Apply these features to the dataset
def get_features(data_list):
    x = pd.DataFrame(data_list)
    x = (apply_nlcs(fuzzysimilarity(x)))
    x = (ngram_features(x))
    x = (nwmd(x))
    #x = (generateoverlaps.syntactics(x))
    return x
```

In [8]:
```
# X - Y split and scaling
import pandas as pd
from sklearn.preprocessing import StandardScaler


def x_y_split(df):
    Y = pd.DataFrame(df[2])
    df.drop(2, inplace=True, axis=1)
    return df, Y


def preprocess_ds(df):
    df.drop(1, inplace=True, axis=1)
    df.drop(0, inplace=True, axis=1)
    return df


def scale_df(df):
    scaler = StandardScaler()
    v = df.values.tolist()
    df_scaled = pd.DataFrame(scaler.fit_transform(v))
    return df_scaled


def dsSplitter(l):
    x = get_features(l)

    # At this stage we have our dataset
    x = preprocess_ds(x)

    # Scaling the dataset
    X, Y = x_y_split(x)
    X = pd.DataFrame(scale_df(X))
    y = Y.values.tolist()
    Y = []
    for i in y:
        Y.append(i[0])
    return X, Y
```

In [9]:
```
X, Y = dsSplitter(y)
```

In [10]:
```
X
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9
0	0.249096	-0.202392	-0.368471	0.154474	-0.021533	0.108093	-0.268702	-0.291004	-0.145425	-0.315959
1	-1.273308	0.554983	-0.069904	-1.065134	0.013835	-0.045936	1.100659	0.419766	-0.067558	-0.369964
2	-0.391916	0.554983	-0.163206	-0.049709	-1.292382	-1.509848	0.244808	-1.238698	-1.292003	0.663582
3	-1.834194	0.554983	-1.702688	1.244928	-0.831929	-1.080914	0.387450	-0.859620	-0.977619	-0.326633
4	0.088843	0.554983	0.349954	0.801053	-0.046452	-0.523299	1.100659	-0.072306	-0.483586	0.336392
...	...	...	...	...	...	...	...	...	...	...
49396	0.249096	0.554983	1.257854	-1.289390	-0.883005	-0.636047	0.442312	-0.909065	-0.713536	0.604907
49397	-0.552169	0.554983	-0.603058	0.404031	-1.209320	-1.772928	0.489337	-1.238698	-1.501593	-0.148548
49398	-1.353435	0.302525	-1.469433	-0.366659	-0.644987	-1.432472	0.322613	-0.859620	-1.321945	0.164255
49399	-2.635460	0.554983	-2.272866	-2.272667	-2.196686	-1.096134	-3.178594	-1.897963	-1.045963	2.570832
49400	-0.071410	0.302525	-0.518471	0.415802	-0.269779	-0.244087	0.113139	-0.458244	-0.464676	-0.017053

49401 rows × 10 columns

In [11]:
```
X1, Y1 = dsSplitter(y1)
```

In [12]:
```python
# The machine learning part

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier


def train_lr(Xtrain, Ytrain):
    lrModel = LogisticRegression(max_iter=10000)
    lrModel.fit(Xtrain, Ytrain)
    return lrModel


def rfModel(Xtrain, Ytrain):
    rf = RandomForestClassifier()
    rf.fit(Xtrain, Ytrain)
    return rf
```

In [13]:
```python
# The actual training part, where the models are trained using the training data

lrModel = train_lr(X, Y)
rfModel = rfModel(X, Y)
```

In [14]:
```python
Y2 = lrModel.predict(pd.DataFrame(X1))

Y3 = rfModel.predict(pd.DataFrame(X1))

X2 = X1.values.tolist()
for i in range(0, len(y1)):
    # Obvious lazy plagiarism is sometimes missed.
    if y1[i][0] == y1[i][1]:
        Y2[i] = 1
        Y3[i] = 1

print("Logistic Regression Accuracy: ", metrics.accuracy_score(Y1, Y2))
print("Logistic Regression Precision: ", metrics.precision_score(Y1, Y2))
print("Logistic Regression Recall: ", metrics.recall_score(Y1, Y2))
print("Logistic Regression F1 score: ", metrics.f1_score(Y1, Y2))
print(" ")
print("Random Forest Accuracy: ", metrics.accuracy_score(Y1, Y3))
print("Random Forest Precision: ", metrics.precision_score(Y1, Y3))
print("Random Forest Recall: ", metrics.recall_score(Y1, Y3))
print("Random Forest F1 score: ", metrics.f1_score(Y1, Y3))
```

```
Logistic Regression Accuracy:  0.7215
Logistic Regression Precision:  0.7235133287764867
Logistic Regression Recall:  0.5986990950226244
Logistic Regression F1 score:  0.6552151036830702

Random Forest Accuracy:  0.77
Random Forest Precision:  0.7872628726287263
Random Forest Recall:  0.6572398190045249
Random Forest F1 score:  0.716399506781751
```

5. RESULTS

Using the above model for lexical and semantic analysis, we obtained an accuracy of 77% using the Random Forest Classifier. This is very close to the results obtained in [7], since in this case, a single classifier is able to produce results very close to that with three classifiers together in [7]. This also shows that our model has a slightly better accuracy than the model in [7]. However, for logistic regression, our model reports a lesser accuracy than the model in [7].

However, the project code is currently not fully functional. The absence of an effective PDF parser is a bottleneck for the completion of the project. We explored pdfminer, PyPDF2, and other repositories using these libraries for python PDF parsers, but they were either non-functional, or did not parse the text correctly. Consequently, the web mining component was not tested enough to verify its accuracy in providing pre-processed text from online sources. That part is still under active development. Therefore, in this demonstration, we will compare the accuracies of the models. A corresponding Jupyter Notebook has also been attached to show that we were able to obtain 77% accuracy using the Random Forest Classifier alone.

Comparison between our proposed model and the model proposed in [7]. These experiments were conducted in a Linux Mint 20 64-bit virtual machine with the host running Windows 11 using 8-core AMD Ryzen 4000 series CPU and 10.5 GB memory allocated to the Virtual Machine:

	[7]	**Proposed model (Random Forest)**
Accuracy (%)	77.10 [7]	**77.0**
Precision (%)	79.42 [7]	**78.7**
Recall (%)	88.49 [7]	**65.72**
F1-score (%)	83.71 [7]	**71.63**
Time taken (s)	Kernel failure	**144.84**
Maximum CPU temperature (℃)	89.5	**85.0**
Maximum power consumption (W)	25.9	**26.3**

6. CONCLUSION

Once completed and ready for deployment, this plagiarism detection tool can help in detecting plagiarized sentences out of various documents including project reports, research papers, journal articles, and class assignments – all for free. This tool will also detect paraphrases so that dishonest students who use paraphrasing tools who still return the same plagiarized text in a paraphrased format are awarded the same plagiarism score.

7. REFERENCES

[1] Empower Students to Do Their Best, Original Work | Turnitin (https://turnitin.com)

[2] Pricing | iThenticate (https://ithenticate.com/pricing)

[3] Free Online Plagiarism Checker - Check Plagiarism (check-plagiarism.com)

[4] Plagiarism Checker - 100% Free Online Plagiarism Detector (smallseotools.com)

[5] Sakamoto, Daisuke & Tsuda, Kazuhiko. (2019). A Detection Method for Plagiarism Reports of Students. Procedia Computer Science. 159. 1329-1338. 10.1016/j.procs.2019.09.303.

[6] Khiled, Farah & Al-Tamimi, Mohammed. (2021). Hybrid System for Plagiarism Detection on A Scientific Paper. Turkish Journal of Computer and Mathematics Education (TURCOMAT). 12. 5707-5719.

[7] Saha, Rudradityo & Kumar, G.. (2021). A Novel Approach for Developing Paraphrase Detection System using Machine Learning. International Journal of Computer Applications. 183. 29-36. 10.5120/ijca2021921389.

[8] GitHub - google-research-datasets/paws: This dataset contains 108,463 human-labeled and 656k noisily labeled pairs that feature the importance of modeling structure, context, and word order information for the problem of paraphrase identification.

[9] Zhang, Y., Baldridge, J., & He, L. (2019). PAWS: Paraphrase Adversaries from Word Scrambling. ArXiv, abs/1904.01130.

[10] GitHub - stanfordnlp/GloVe: GloVe model for distributed word representation

[11] word2vec - Can't load glove.6B.300d.txt - Stack Overflow (https://stackoverflow.com/a/52366212)