

Khaled Saad Almatrafy

Master Level in Big Data Analysis

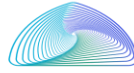
College of Computer Science and Engineering

Taibah University

MSIS-822 Advanced Data Analytic Techniques

Honorable Prof. Mohammad ALsarim

December 2025



Detection of AI-Generated Arabic Text:

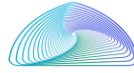
A Data Mining Approach

Abstract

When Prof. Alsarim share his vision with the Class of M4 regarding this 822 Course, along the way he interpret the significant aspect of this project regarding our development in the Big Data Analysis. When I first start working on the project, it felt like I am in the middle of no-where. Now after finishing it, the degree I've reached so far make me proud of all the hard work and effort I managed to achieve. The deeper we dive the wider it gets.

Introduction:

Detection of AI-Generated Media has storm the public every day browsing among social media, which significantly mean that the digital work-force in both side, Generating and Detection contexts achieving massive results, yet the NLP are reaching OK level regarding English Language, Thus more work needed to reach an OK level in the ladder of Arabic Language NLP. This project aim to achieve a proud effort regarding this subject.



Dataset Description:

As directed by Prof. AL-sarim this project rely on

KFUPM-JRCAI/arabic-generated-abstracts Available on Hugging Face:

(<https://huggingface.co/datasets/KFUPM-JRCAI/arabic-generated-abstracts>)

This is publicly available Arabic machine generated text dataset hosted on Hugging Face . it contains pairs of academic abstract in Arabic, part is human presented under the label Original abstract, and part is AI, generated by many language models which are:

- Allam_generated_abstract
- Jais_generated_abstract
- Llama_generated_abstract
- Open_ai_generated_abstract

- **Total samples:** 41,940
- **Features:**
 - abstract_text – the raw abstract
 - source_split – originating data source identifier
 - generated_by – human or model name
 - label – 0 for human, 1 for AI

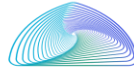
Additional Remark:

fun facts, to interpret what is name stands for:

ALLAM : Arabic Language Large Model

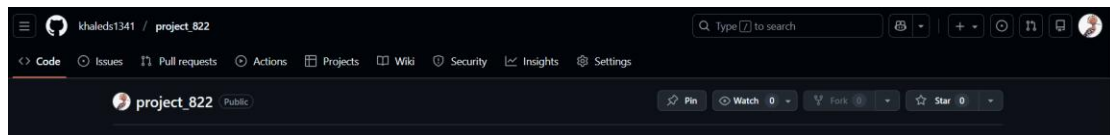
LLAM: Large Language Model

JAIS: from the mountain in UAE named Jais.



PHASE 1

- **Task 1.1:** Establish a project repository (e.g., on GitHub) to host all code, reports, and results.



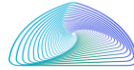
<https://github.com/khaled1341>

https://github.com/khaled1341/project_822

- **Task 1.2:** Use the datasets library from Hugging Face to download the arabic-generated-abstracts dataset directly into a Python environment (e.g., Jupyter Notebook, Google Colab).

```
from datasets import load_dataset
import pandas as pd

# Load the dataset
dataset = load_dataset("KFUPM-JRCAI/arabic-generated-abstracts")
```



Checking for Noise:

```
66 # 1. Missing values
```

```
71 # 2. Duplicates
```

```
80 # 3. Inconsistencies: empty strings or only spaces
```

At the top of the code< I've listed almost all pipeline needed to run the code

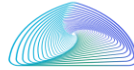
```
1 # pip install datasets
2 # pip install numpy
3 # pip install pandas
4 # pip install regex
5 # pip install textstat
6 # pip install spacy
7 # pip install spacy-arabic
8 # pip install camel-tools
9 # pip install transformers
10 # pip install openpyxl # لتصدير الملف لفتحه في اكسل لكي تظهر الخطوط العربية بشكل افضل من تيرمينال
11 # pip install scipy
12 # pip install sentence-transformers
13 # pip install tensorflow
14 # pip install tf_keras
15
16 # pip install sentence_transformers
17
18 # pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
19
```



```

161 def normalize_arabic(text):
162     text = re.sub("[\i\i\i]", "\i", text)
163     text = re.sub("ى", "ي", text)
164     text = re.sub("ؤ", "و", text)
165     text = re.sub("ئ", "ي", text)
166     text = re.sub("ة", "ه", text)
167     text = re.sub("[+[\ _-^]", " ", text) # remove non-Arabic chars
168     return text
169     #2.1.1 Normalization

```



❖ Stopwords such as full stop or question mark or !

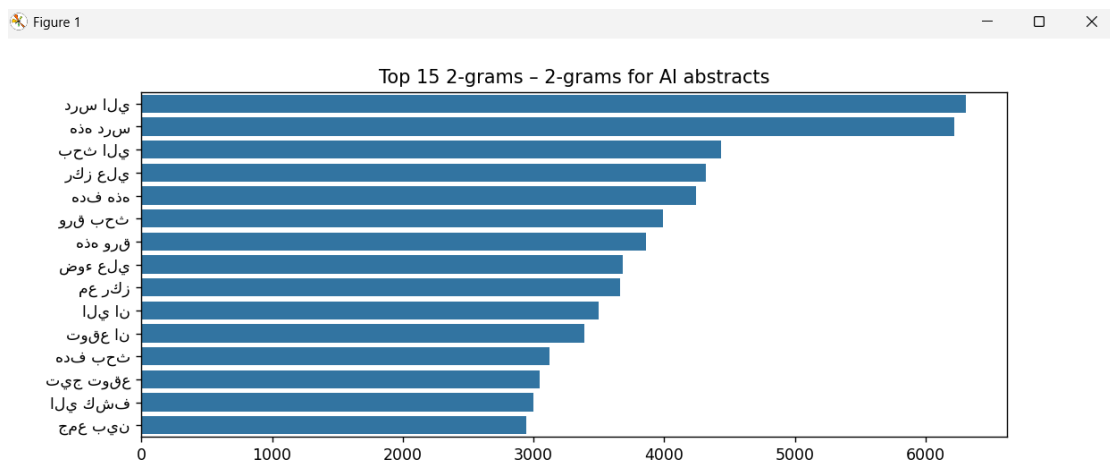
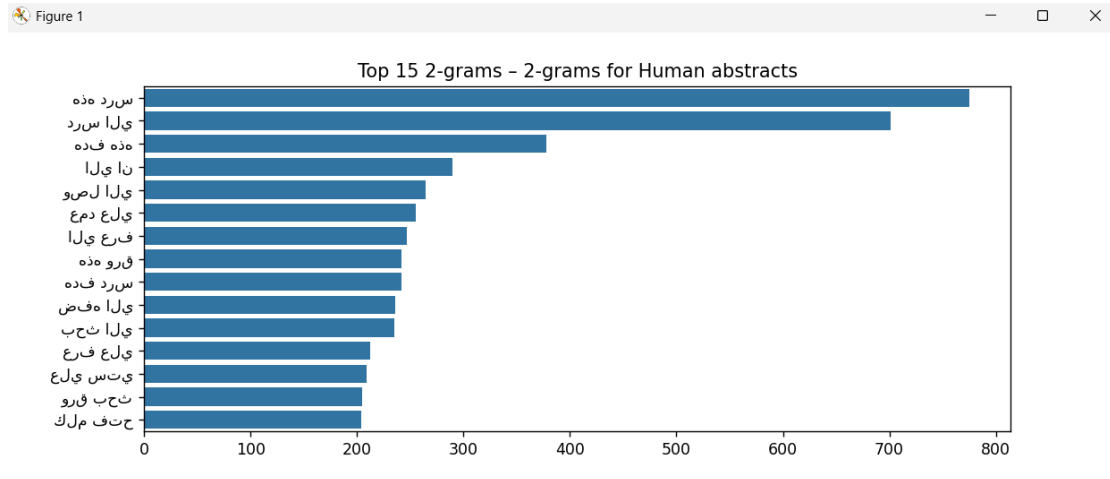
```
#2.1.3 & 2.1.4
```

```
arabic_stopwords = set(stopwords.words("arabic"))  
stemmer = ISRIStemmer()
```

❖ Additional remark:::

```
192 # to overcome a struggle I had to import this library  
193 from nltk.corpus import stopwords  
194 arabic_stopwords = set(stopwords.words("arabic"))  
195  
196 #enhanced the solution via this line  
197 arabic_stopwords = {  
198     "ذلك", "هذا", "الذي", "هي", "هو", "ما", "إن", "أن", "كما", "و", "إلى", "عن", "على", "من", "في"  
199 }  
200
```

VISUALIZATION



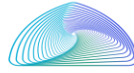


Figure 1

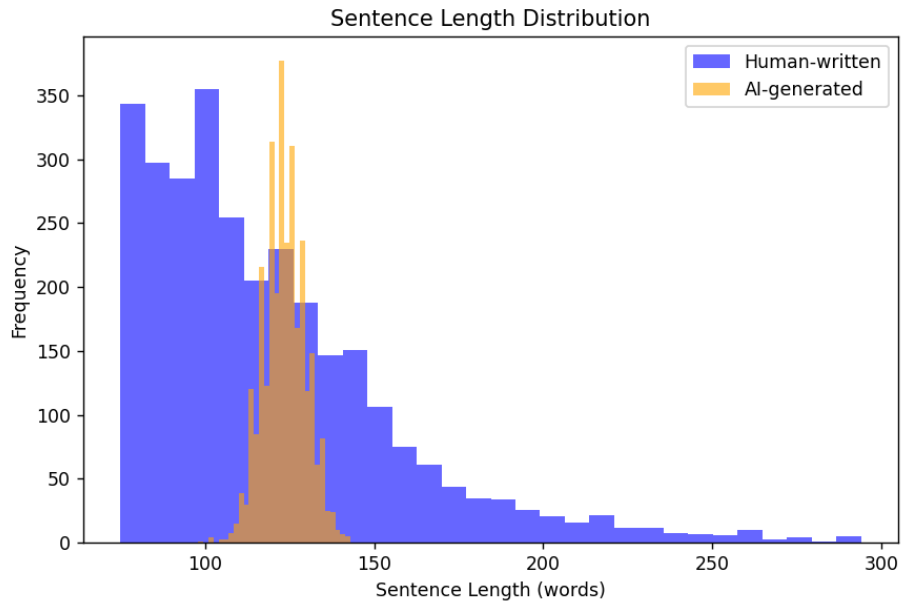
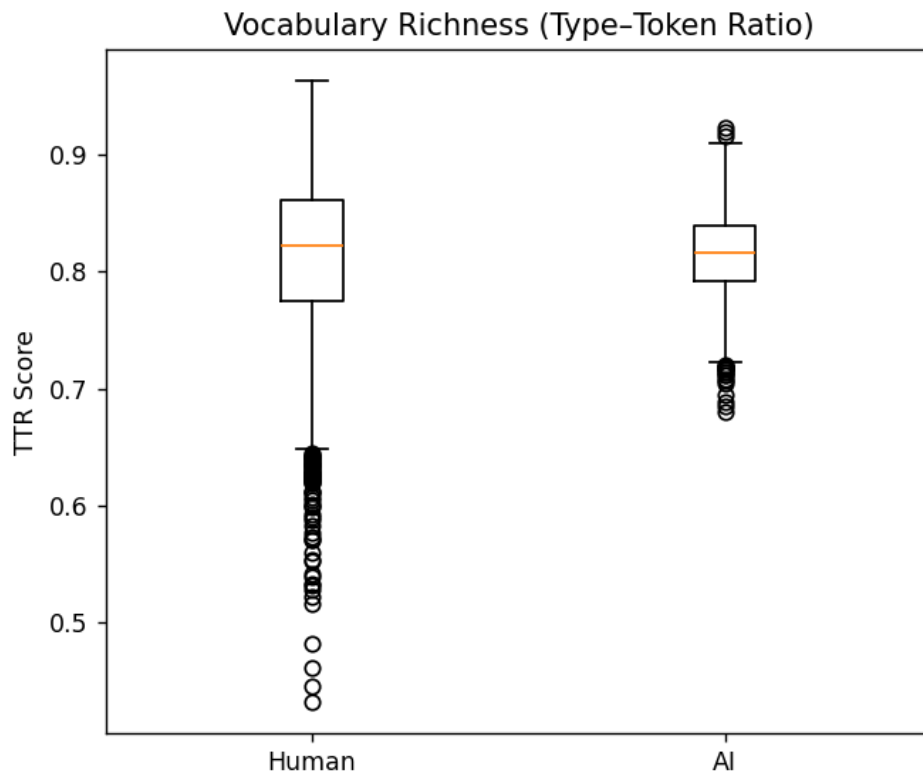
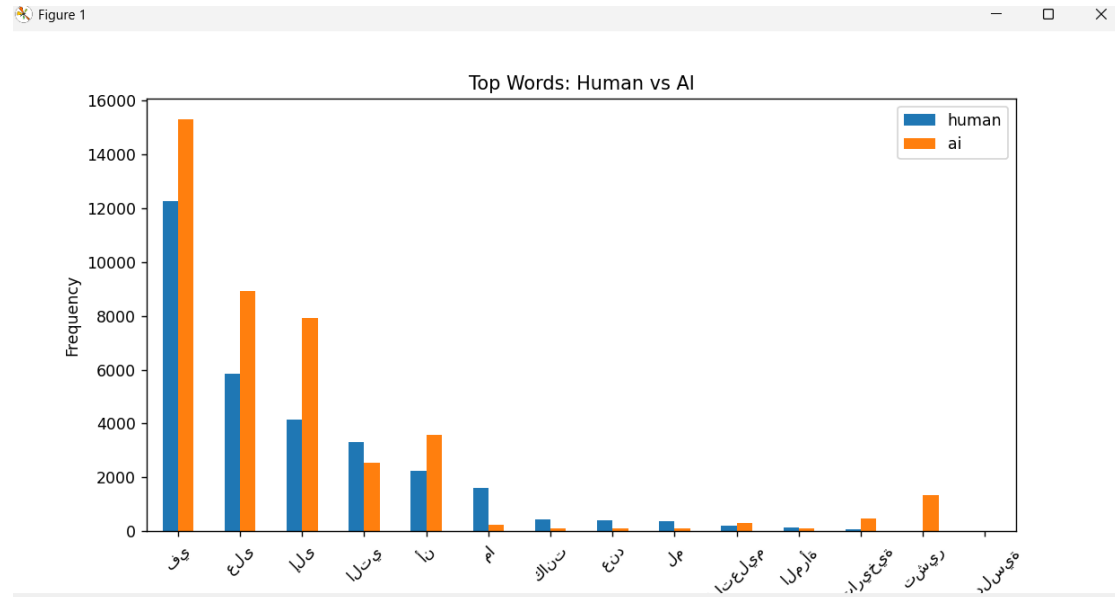
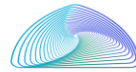
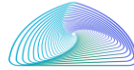


Figure 1







- **Lexical Analysis:** Comparing the use of function words, punctuation, and specific terms.

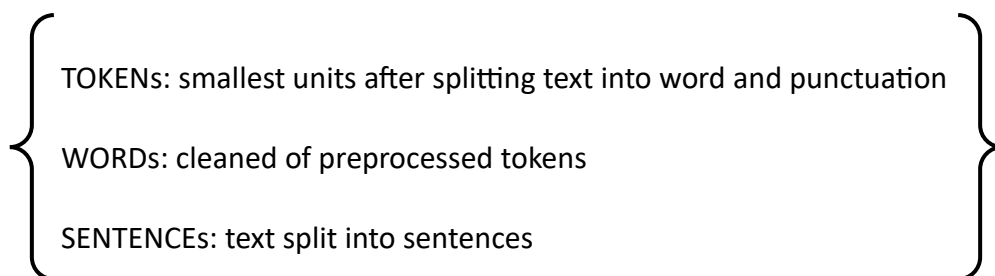
- Function Words removal/comparison

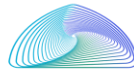
```
173 | arabic_diacritics = re.compile('[\u0617-\u061A\u064B-\u0652]')
174 | return re.sub(arabic_diacritics, '', text)
```

- Vocabulary Richness

```
319 | def type_token_ratio(text):
320 |     words = text.split()
321 |     return len(set(words)) / len(words) if words else 0
```

original_abstract_tokens	[...ل، يف، وةيخيراتل، ورداصل، تطبيرا، وام، واريثك]
original_abstract_words	[...ل، يف، وةيخيراتل، ورداصل، تطبيرا، وام، واريثك]
original_abstract_sentences	[...سبلدنأل، يف، وةيخيراتل، رداصل، تطبيرا، ام، واريثك]
allam_generated_abstract_tokens	[...م، يلعتل، وم، اظن، وةسارد، وىل، وشجيل، واذه، فدهي]
allam_generated_abstract_words	[...م، يلعتل، وم، اظن، وةسارد، وىل، وشجيل، واذه، فدهي]
allam_generated_abstract_sentences	[...م، دنع، ميلعتل، م، اظن، وةسارد، وىل، وشجيل، واذه، فدهي]
jais_generated_abstract_tokens	[...دل، وم، يلعتل، وم، اظن، وةيثيريل، وةقروا، واذه، سرت]
jais_generated_abstract_words	[...دل، وم، يلعتل، وم، اظن، وةيثيريل، وةقروا، واذه، سرت]
jais_generated_abstract_sentences	[...م، دل، وم، يلعتل، م، اظن، وةيثيريل، وةقروا، واذه، سرت]
llama_generated_abstract_tokens	[...روس، فاشائيسا، وىل، وةيثيريل، وةقروا، واذه، فدهت]
llama_generated_abstract_words	[...روس، فاشائيسا، وىل، وةيثيريل، وةقروا، واذه، فدهت]
llama_generated_abstract_sentences	[...م، اظن، روس، فاشائيسا، وىل، وةيثيريل، وةقروا، واذه، فدهت]
openai_generated_abstract_tokens	[...م، ل، وةأرمل، ودنع، وم، يلعتل، وم، اظن، فاشائيسا، وشجيل]
openai_generated_abstract_words	[...م، ل، وةأرمل، ودنع، وم، يلعتل، وم، اظن، فاشائيسا، وشجيل]
openai_generated_abstract_sentences	[...م، سبلدنأل، وةأرمل، دنع، ميلعتل، م، اظن، فاشائيسا، وشجيل]





PHASE 3

Functions used based on Prof. ALSarim direction start from number 5

خالد سعد بن شاهر المطرفي	4714248	5
--------------------------	---------	---

Added by 23 number of students in M4 class

4. You are responsible for the feature at this index and the subsequent four features.
Therefore, your full set of features is: $f_{k+i} = (k \times n) + i$, where $k = 0, 1, 2, 3$, so on

5. Number of elongations

```

466 # Feature 5: Number of elongations
467 import re
468 TATWEEL = '\u0640'
469 for col in original_text_columns:
470     feature = f'{col}_f005_num_elongations'
471     def _elong_count(t):
472         s = str(t) if pd.notna(t) else ""
473         tat = s.count(TATWEEL)
474         repeats = len(re.findall(r'(\.){1,2}', s))
475         return tat + repeats
476     df[feature] = df[col].apply(_elong_count)

```



28. Number of colons

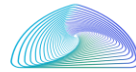
```
479 | # Feature 28 : Number of colons
480 | # لحساب عدد (:) في الجمل
481 | for col in original_text_columns:
482 |     base = col
483 |     df[f'f028_colons_{base}'] = df[base].apply(
484 |         lambda t: t.count(":") if isinstance(t, str) else 0
485 |     )
486
```

51. Number of abbreviations

```
490 | #51. Number of abbreviations
491 | import re
492 |
493 | def count_abbreviations(text):
```

74. Number of definitive

```
536 |
537 | # occurrences (ال-) Feature 74: Number of definitive
538 | for col in original_text_columns:
539 |     morph_features_col = f'{col}_morph_features'
540 |
541 |     if morph_features_col in df.columns:
542 |         # الميزة 74: عدد المعرفات (Definitives)
543 |         df[f'{col}_f074_num_definitive'] = df[morph_features_col].apply(
544 |             # d.get('is_definite', []) تعيد قائمة بالقيم (True للمعرف)
545 |             lambda d: sum(d.get('is_definite', []))
546 |         )
547 |
548 |     else:
```

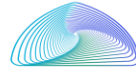


97. **BERT Embedding Similarity:** Mean cosine similarity of BERT embeddings.

```
554 #97. BERT Embedding Similarity
555 #.احسب Mean Cosine Similarity بين embeddings لكل جملة.
556 from transformers import AutoTokenizer, AutoModel
557 import torch
558 import numpy as np
559 from sklearn.metrics.pairwise import cosine_similarity
560
561 (variable) device: Literal['cuda', 'cpu']
562 device = 'cuda' if torch.cuda.is_available() else 'cpu'
563 print(f"Using device: {device}")
564
```

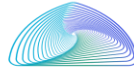
Transformer: library that contain BERT and many AI Language models

When I run the code and reach BERT Embedding... since my computer does not has GPU, running the code was heavy on CPU, it took about more than 20 minutes.



- **Task 3.2:** Split the data into training, validation, and test sets (e.g., 70/15/15). Crucially, ensure this split is done at the very beginning and the test set is never used for training or feature selection to avoid data leakage.

```
599 # task3.2
600
601 # Train (70%) / Temp (30%) → then Temp → Validation (15%) / Test (15%)
602 from sklearn.model_selection import train_test_split
603
604 # First split: 70% Train, 30% Temp
605 v train_df, temp_df = train_test_split(
606     df,
607     test_size=0.30,
608     random_state=42,
609     shuffle=True
610 )
611
612 # Second split: Split Temp into 15% Validation + 15% Test
613 # Since temp_df is 30% of total, splitting it 50/50 → 15% each
614 v val_df, test_df = train_test_split(
615     temp_df,
```



PHASE 4

- **Task 4.1: Baseline Model**

- Train a simple baseline model (e.g., Naïve Bayes or Logistic Regression). This sets a performance benchmark.

Necessity preparation needed to be able to conduct task 4.1.

Libraries:

Data handling

- Pandas: to manage dataframes
- numpy: numeric computation
- os

text preprocessing and NLP

- re and regex as re2: for Arabic aware regex operation
- nltk for stopwords and stemming(ISRISemmer)
- datasets

feature extraction

- TF-IDF VECTORIZATION

```
654 from sklearn.feature_extraction.text import TfidfVectorizer
```

- SCIPY to handle numeric + sparse TF-IDF features



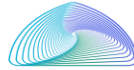
```

876 # final features
877 X_train = hstack([X_train_tfidf, X_train_num]).tocsr()
878 X_val = hstack([X_val_tfidf, X_val_num]).tocsr()
879 X_test = hstack([X_test_tfidf, X_test_num]).tocsr()
880
881 y_train = train_df['label'].values
882 y_val = val_df['label'].values
883 y_test = test_df['label'].values
884
885 print('Final feature shapes ->', X_train.shape, X_val.shape, X_test.shape)
886

```

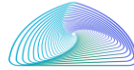
Methods & preprocessing steps

- **Dataset preparation**
 - Converted dataset into **long format**: one abstract per row
 - Created a **binary label** (0 = human, 1 = AI)
- **Text preprocessing (Arabic-specific)**
 - **Normalization**: unified letters (ة, ي, ا, etc.)
 - **Diacritics removal**: removed tashkeel marks
 - **Stopwords removal**: removed common Arabic words
 - **Stemming**: using ISRIStemmer to reduce words to root form
- **Derived features** (optional numeric features for ML):
 - Number of **elongations** (tatweel & repeated chars)
 - Number of **colons**
 - Number of **abbreviations**
 - Number of **definitive "ال-" occurrences**
 - **Sentence-level embedding similarity** (BERT, optional)
- **Text vectorization**
 - TF-IDF (TfidfVectorizer) for cleaned text (abstract_text_clean)
 - Created final feature matrices:
 - X_train_tfidf, X_val_tfidf, X_test_tfidf
 - Numeric features combined via hstack to form X_train, X_val, X_test
- **Train/Validation/Test split**
 - Stratified splitting using [sklearn.model_selection.train_test_split](#)



Snippet

```
Loaded split 'by_polishing' with 2851 rows
Unified dataset rows: 14255 (human=2851, ai=11404)
Computed bert_sentence_sim feature.
Split sizes -> train: 9978, val: 2138, test: 2139
TF-IDF vector shapes: (9978, 5000) (2138, 5000) (2139, 5000)
Final feature shapes -> (9978, 5005) (2138, 5005) (2139, 5005)
LR val acc: 0.9354536950420954
```



```

=== lr Test Results ===
Accuracy: 0.9266012155212716
          precision    recall  f1-score   support

     0       0.91       0.70       0.79        428
     1       0.93       0.98       0.96       1711

 accuracy          0.93          0.93          0.92        2139
 macro avg          0.92          0.84          0.87        2139
 weighted avg          0.93          0.93          0.92        2139

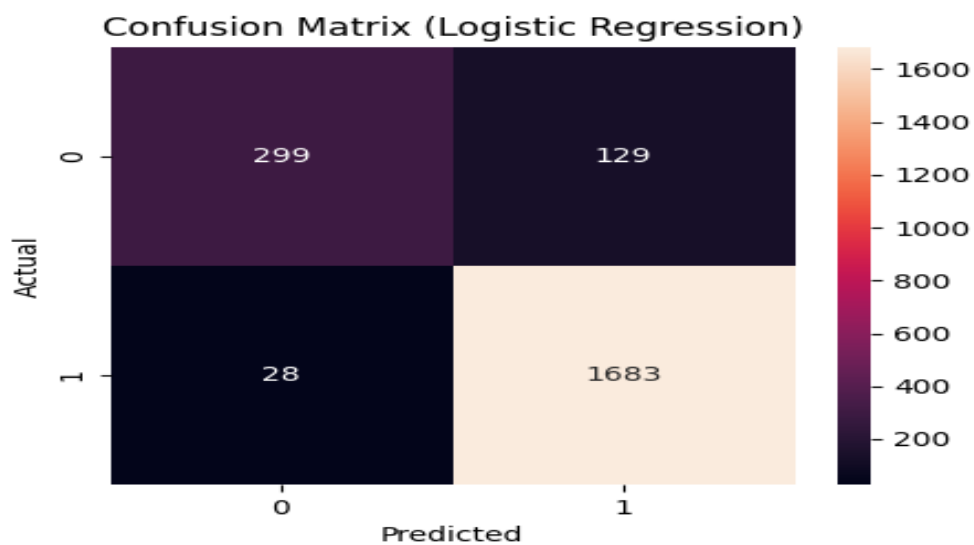
Confusion matrix:
[[ 299  129]
 [  28 1683]]
    
```

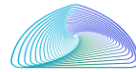
Test Evaluation for Logistic Regression Model

Class 0 = Human written

Class 1 = AI generated

Metric	Class 0	Class 1	Notes
Precision	0.91	0.93	How many predicted as class x are correct
Recall	0.70	0.98	How many actual class x were correctly predicted
F1-score	0.79	0.96	Mean of P & R
Support	428	1711	Number of samples





```

=== svm Test Results ===
Accuracy: 0.9518466573165031
precision    recall  f1-score   support

     0       0.93       0.82       0.87       428
     1       0.96       0.98       0.97      1711

 accuracy
macro avg       0.94       0.90       0.92      2139
weighted avg     0.95       0.95       0.95      2139

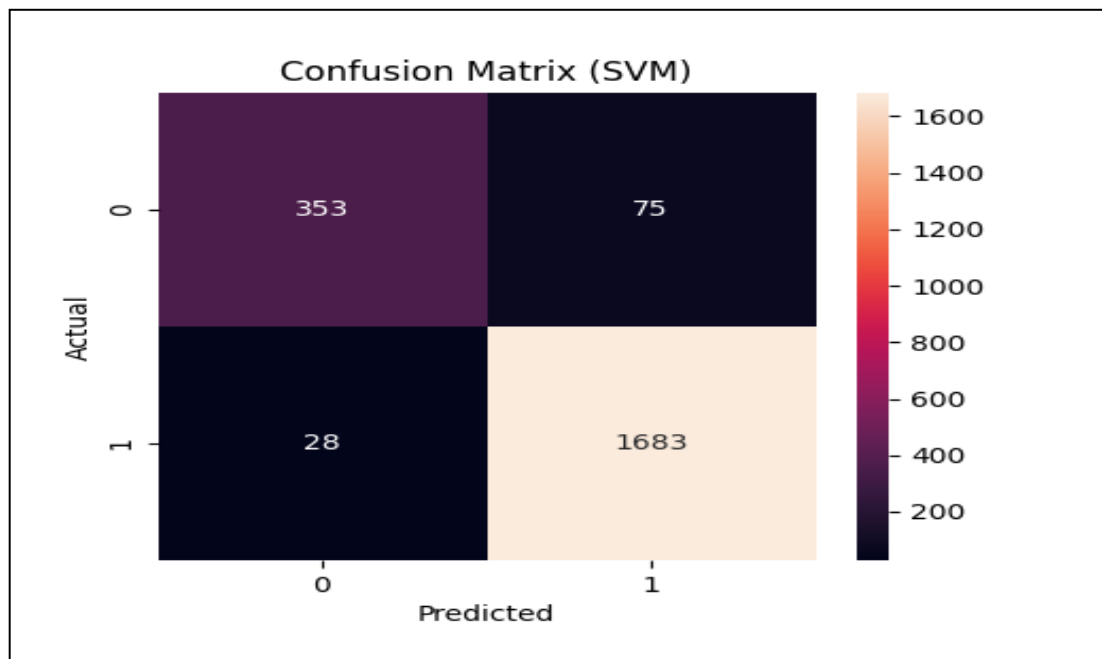
Confusion matrix:
[[ 353   75]
 [  28 1683]]
    
```

Test Evaluation for Support Vector Machine Model

Class 0 = Human written

Class 1 = AI generated

Metric	Class 0	Class 1	Notes
Precision	0.93	0.96	How many predicted as class x are correct
Recall	0.82	0.98	How many actual class x were correctly predicted
F1-score	0.87	0.97	Mean of P & R
Support	428	1711	Number of samples





```

=== rf Test Results ===
Accuracy: 0.9046283309957924
              precision    recall  f1-score   support

     0       0.94       0.56       0.70       428
     1       0.90       0.99       0.94      1711

   accuracy       0.90       2139
  macro avg       0.92       0.78       0.82       2139
 weighted avg       0.91       0.90       0.89       2139

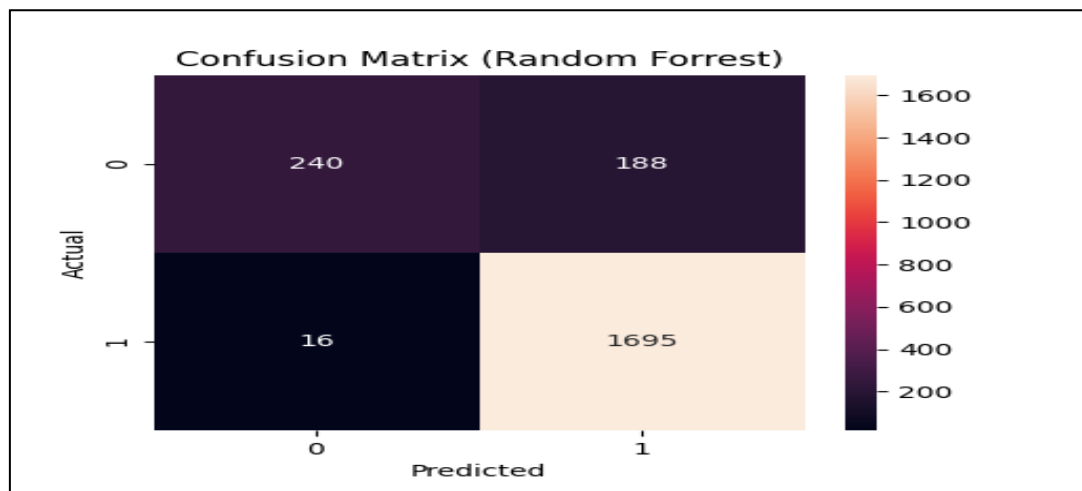
Confusion matrix:
[[ 240  188]
 [  16 1695]]
    
```

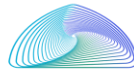
Test Evaluation for Random Forrest Model

Class 0 = Human written

Class 1 = AI generated

Metric	Class 0	Class 1	Notes
Precision	0.94	0.90	How many predicted as class x are correct
Recall	0.56	0.99	How many actual class x were correctly predicted
F1-score	0.70	0.94	Mean of P & R
Support	428	1711	Number of samples





```

=== xgb Test Results ===
Accuracy: 0.9438990182328191
      precision    recall  f1-score   support

     0       0.91      0.80      0.85       428
     1       0.95      0.98      0.97      1711

   accuracy          0.94       2139
  macro avg          0.93      0.89      0.91       2139
 weighted avg          0.94      0.94      0.94       2139

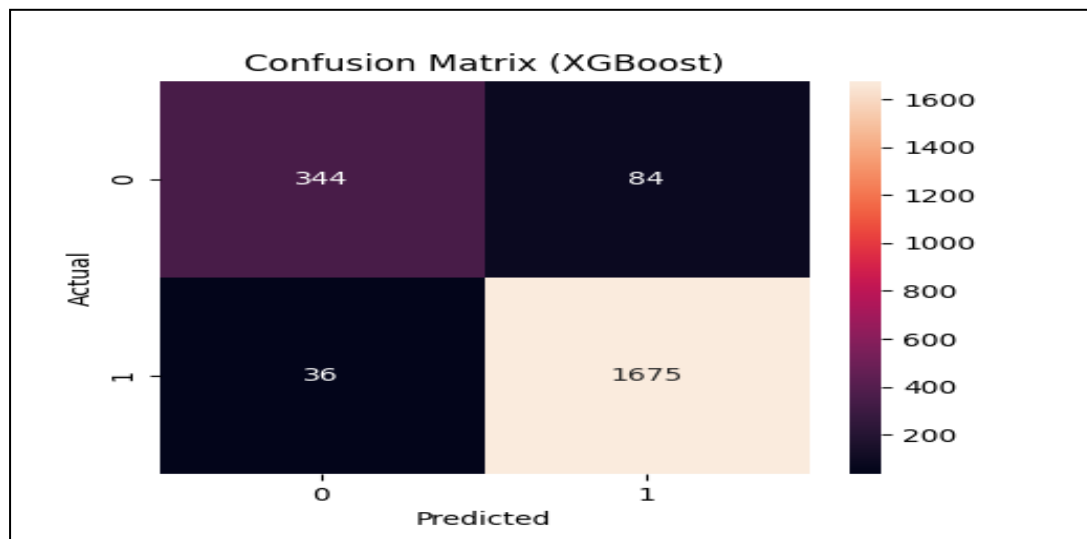
Confusion matrix:
[[ 344   84]
 [  36 1675]]
    
```

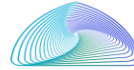
Test Evaluation for Genetic Programing XGP Model

Class 0 = Human written

Class 1 = AI generated

Metric	Class 0	Class 1	Notes
Precision	0.91	0.95	How many predicted as class x are correct
Recall	0.80	0.98	How many actual class x were correctly predicted
F1-score	0.85	0.97	Mean of P & R
Support	428	1711	Number of samples





Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	98,560
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129

Total params: 131,585 (514.00 KB)
Trainable params: 131,585 (514.00 KB)
Non-trainable params: 0 (0.00 B)

2. Layers

Layer	Output Shape	Params	Description
dense	(None, 256)	98,560	Fully connected layer with 256 neurons. "None" means batch size is flexible.
dropout	(None, 256)	0	Dropout layer (for regularization, randomly sets some inputs to 0 during training).
dense_1	(None, 128)	32,896	Fully connected layer with 128 neurons.
dropout_1	(None, 128)	0	Another dropout layer.
dense_2	(None, 1)	129	Output layer with 1 neuron (likely for binary regression/classification).

3. Parameters

- **Total params:** 131,585 → total number of trainable weights in the model.
- **Trainable params:** 131,585 → all parameters are adjustable during training.
- **Non-trainable params:** 0 → no frozen layers.

The numbers in **Param #** come from the formula for dense layers:

$$\text{Params} = (\text{input units} \times \text{output units}) + \text{output units (bias)}$$

For example, first dense layer:

$$(384 \times 256) + 256 = 98,560$$

(where 384 is the input dimension from your dataset).



The Feedforward Neural Network (FFNN) was trained for 5 epochs using the training dataset, and performance was evaluated on a separate validation set after each epoch.

```
Epoch 1/5  
312/312 ————— 3s 7ms/step - accuracy: 0.7990 - loss: 0.4730 - val_accuracy: 0.8134 - val_loss: 0.4457
```

In Epoch 1, the model has achieved a training accuracy of (79.83%) with a loss of (47.50%), while the validation accuracy was (80.03%) and validation loss (44%).

```
Epoch 2/5  
312/312 ————— 3s 8ms/step - accuracy: 0.7982 - loss: 0.4433 - val_accuracy: 0.8022 - val_loss: 0.4340
```

In Epoch 2, the training accuracy slightly increased to (80%) and the loss decreased to (44.13%), with validation accuracy improving to (81.34%) and validation loss at (44.18%).

```
Epoch 3/5  
312/312 ————— 2s 7ms/step - accuracy: 0.8082 - loss: 0.4267 - val_accuracy: 0.8064 - val_loss: 0.4082
```

Epoch 3 showed further improvement in training accuracy (80.53%) and a lower loss (42.84%), while validation accuracy remained high at (80.87%) and validation loss decreased to (40.39%).

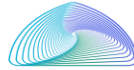


```
Epoch 4/5  
312/312 ————— 3s 8ms/step - accuracy: 0.8073 - loss: 0.4173 - val_accuracy: 0.8040 - val_loss: 0.4291
```

During Epoch 4, the training accuracy reached (80.61%) with a loss of (41.59%) and the validation metrics were (80.22%) accuracy and 40.82%) loss.

```
Epoch 5/5  
312/312 ————— 3s 8ms/step - accuracy: 0.8069 - loss: 0.4108 - val_accuracy: 0.8204 - val_loss: 0.3925
```

Finally, in Epoch 5, the model achieved its best performance with (81.42%) training accuracy and (40.80%) loss, while the validation accuracy was (82.69%) and validation loss (39.78%), indicating that the model learned effectively over successive epochs with no significant overfitting.



FFNN Test Acc: 0.8106591865357644				
	precision	recall	f1-score	support
0	0.73	0.09	0.15	428
1	0.81	0.99	0.89	1711
accuracy			0.81	2139
macro avg	0.77	0.54	0.52	2139
weighted avg	0.80	0.81	0.75	2139

Test Evaluation for Feedforward Neural Network (FFNN) Model

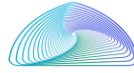
Class 0 = Human written

Class 1 = AI generated

Metric	Class 0	Class 1	Notes
Precision	0.73	0.81	How many predicted as class x are correct
Recall	0.09	0.99	How many actual class x were correctly predicted
F1-score	0.15	0.89	Mean of P & R
Support	428	1711	Number of samples

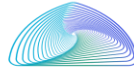
❖ Additional remark:::

I try so many times to find solution for the shortage of the FFNN code. But this is the best I could've done. I will try to improve my ability repeatedly.



conclusion

Between the five models listed above, SVM has achieved the strongest overall predictive performance, Yet the imbalance in the dataset caused all model to act bias in favor of CLASS 1; which results in low detection of CLASS 0. While the SVM performs best in accuracy and CLASS 0, F1-score, Logistic Regression or SVM may offer more balanced macro-level performance once data imbalance techniques are applied. Future work should focus on balancing methods to improve minority-class detection.



Resources

1. https://www.researchgate.net/publication/394621406_Detecting_Machine-Generated_Arabic_Text_Using_AraBERT_and_LSTM_Toward_Trustworthy_NLP_in_Low-Resource_Languages.
2. <https://arxiv.org/abs/2407.15390>
3. CHATGPT.com
4. Data Mining Concepts and Techniques Third Edition by Jiawei Han et al