# Assignment 2 – Data Analysis and Machine Learning

ICLR 2023: Real vs. LLM-generated Paper Analysis

Name: Dashun Liu

Student ID: s2608687

Course: Data Analysis and Machine Learning (DAML)

Date of Submission: March 25, 2025

## Abstract

This study explores the problem of identifying texts produced by large language models (LLMs) in academic contexts, with a focus on submissions to ICLR 2023. We perform preliminary dataset exploration on a dataset containing both human- and LLM-authored paper entries, examining textual characteristics such as length and lexical diversity. Based on these insights, three machine learning problems are formulated: binary classification to detect LLM-generated entries, multiclass classification to identify the source LLM, and regression to predict the confidence score assigned by a black-box detection model. For each task, we evaluate a variety of models and feature representations—including TF-IDF and Sentence-BERT embeddings—using best practices such as cross-validation and hyperparameter tuning. Our results show that SVMs and XGBoost models demonstrate strong results for classification tasks, while confidence score prediction via regression proves to be a difficult task. These findings indicate that automatic recognition of AI-written research papers is plausible in classification settings but limited in predictive granularity when using opaque scoring systems.

## Introduction & Context

Large Language Models (LLMs) are a class of deep learning models that have made significant advances in natural language processing (NLP) tasks. These models are typically composed of billions or even trillions of parameters and are trained using large-scale textual datasets, enabling them to generate and understand language at a level that closely resembles human language (Wikipedia contributors, 2024). The evolution of LLMs has followed a trajectory from early statistical approaches to neural networks, and eventually to large-scale pre-trained transformer-based architectures. This development has given rise to state-of-the-art models capable of handling increasingly complex language tasks with high degrees of fluency and coherence.

Recent examples of such models include OpenAI’s GPT-4o, released in May 2024, which introduced multimodal capabilities allowing it to process both text and visual inputs, and Meta’s Code Llama, launched in 2023 with a focus on code generation and comprehension. These advancements have led to the broad implementation of LLMs across various sectors. In industry, they are used in customer service applications to provide instantaneous and context-aware responses to users (Society, 2023). In education, LLMs are being employed to deliver personalized tutoring, improving accessibility and individualized learning experiences (Kumar and Zhang, 2023). At the same time, their integration into daily workflows has introduced broader societal concerns, including questions about labor displacement, ethical risks, and regulatory oversight (Dojo, 2023).

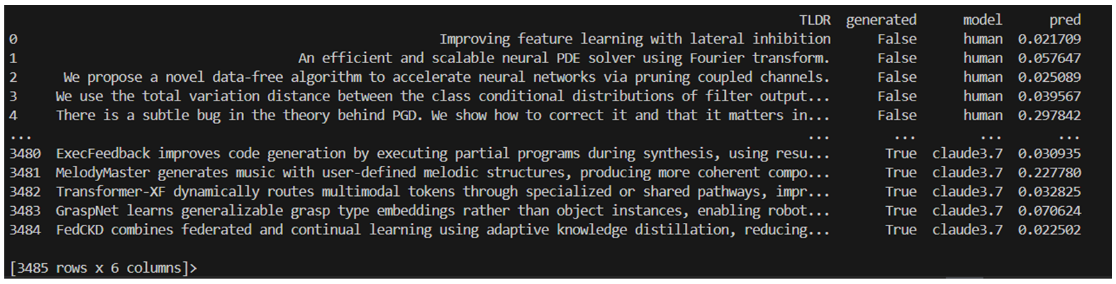
Despite their capabilities, LLMs face significant criticism regarding the reliability and authenticity of their outputs. A well-documented limitation is their tendency to produce hallucinations—outputs that appear linguistically plausible but are factually inaccurate or nonsensical (Simonite, 2023). This issue undermines trust in AI-generated content, especially in high-stakes domains such as scientific research, journalism, and public communication. As

LLMs are increasingly used to draft academic texts, reports, and articles, distinguishing between human-authored and machine-generated content has become a pressing challenge.

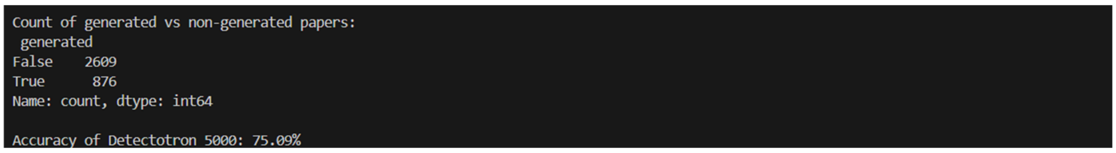
The ability to detect AI-generated text is thus crucial for several reasons. It plays a central role in preserving academic integrity, combating the dissemination of misinformation, and ensuring transparency in automated content creation. Moreover, effective detection mechanisms are essential for enforcing platform-specific policies, such as banning synthetic content in peer-reviewed venues, and for curating high-quality datasets for training future models (Hao, 2023). In response to these concerns, this coursework examines the task of LLM-generated text detection using a dataset of academic paper submissions to ICLR 2023. Through preliminary dataset exploration and the development of machine learning models, this project aims to identify patterns that differentiate human and machine-generated writing, and to evaluate the feasibility of automated detection systems in real-world academic contexts.

## Dataset Summary & Visualizations

The dataset used in this project was constructed by Shiwen Qin (DAML TA) and contains 3,485 entries. Each entry corresponds to either a real academic paper submitted to ICLR 2023 or a synthetic one generated by a large language model (LLM). The dataset consists of six columns: title, abstract, TLDR, generated, model, and pred. These capture, respectively, the title of the paper, its abstract, a one-sentence summary, a binary flag indicating whether the text was generated by an LLM, the specific model used (or "human"), and the predicted probability from the Detectotron 5000 that the entry is fake, in addition, the accuracy of detection is 75.09%.



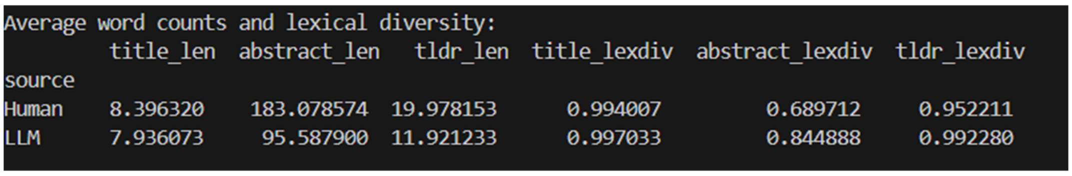
```
                                                                                          TLDR  generated      model      pred
0                                          Improving feature learning with lateral inhibition      False      human  0.021709
1                                 An efficient and scalable neural PDE solver using Fourier transform.      False      human  0.057647
2      We propose a novel data-free algorithm to accelerate neural networks via pruning coupled channels.      False      human  0.025089
3     We use the total variation distance between the class conditional distributions of filter output...      False      human  0.039567
4     There is a subtle bug in the theory behind PGD. We show how to correct it and that it matters in...      False      human  0.297842
...                                                                                            ...        ...        ...       ...
3480  ExecFeedback improves code generation by executing partial programs during synthesis, using resu...       True  claude3.7  0.030935
3481  MelodyMaster generates music with user-defined melodic structures, producing more coherent compo...       True  claude3.7  0.227780
3482  Transformer-XF dynamically routes multimodal tokens through specialized or shared pathways, impr...       True  claude3.7  0.032825
3483  GraspNet learns generalizable grasp type embeddings rather than object instances, enabling robot...       True  claude3.7  0.070624
3484  FedCKD combines federated and continual learning using adaptive knowledge distillation, reducing...       True  claude3.7  0.022502

[3485 rows x 6 columns]>
```



```
Count of generated vs non-generated papers:
 generated
False    2609
True      876
Name: count, dtype: int64

Accuracy of Detectotron 5000: 75.09%
```

**Figure 1 The result of df.head() and the condition about Pred**

Descriptive statistics were computed for several key textual features, including the word counts of the title, abstract, and TLDR, as well as the lexical diversity (unique words / total words) of the abstract. The average abstract length for human-written texts is approximately 183 words, nearly double that of LLM-generated abstracts at around 96 words. A similar trend is observed in TLDRs, where the mean length is ~20 for human texts and ~12 for LLMs. Interestingly, LLM-generated abstracts show higher lexical diversity (mean ≈ 0.84) than those written by humans (mean ≈ 0.69), suggesting more varied vocabulary use — though this may reflect surface-level synonym substitutions rather than genuinely diverse content.



```
Average word counts and lexical diversity:
        title_len  abstract_len   tldr_len  title_lexdiv  abstract_lexdiv  tldr_lexdiv
source
Human    8.396320    183.078574  19.978153      0.994007         0.689712     0.952211
LLM      7.936073     95.587900  11.921233      0.997033         0.844888     0.992280
```

**Figure 2 The average word count and lexical diversity for title, abstract and TLDR**

These patterns (Figure 3) are further visualized using boxplots. Since lexical diversity in titles and TLDRs is relatively similar between human and LLM texts, analysis focuses on the remaining four metrics: abstract_len, title_len, tldr_len, and abstract_lexdiv. Notably, abstract lengths for human-written entries display more variability, with many outliers and a broader range (median ≈ 180 words, max > 350), whereas LLM abstracts tend to cluster tightly around 90 words. For lexical diversity, LLM outputs appear richer on average; however, this may reflect templated phrasing or synonym inflation rather than deeper semantic variety.

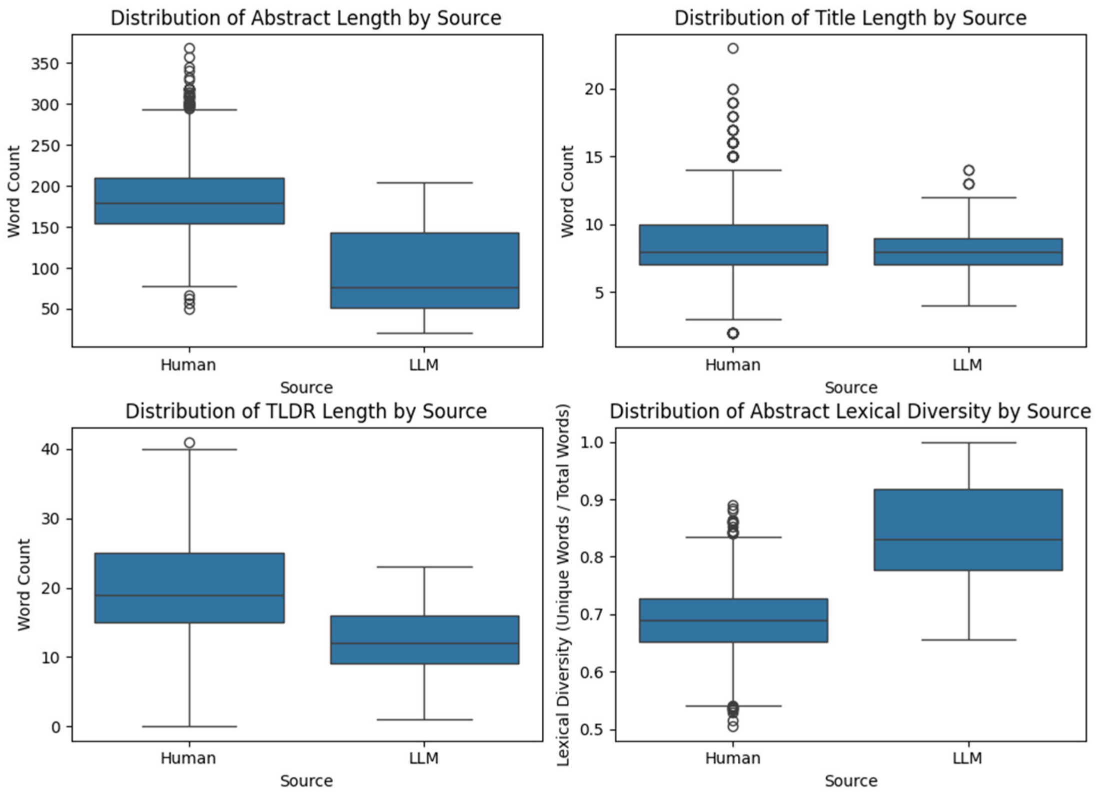


**Figure 3 The boxplots of 'title_len', 'abstract_len', 'tldr_len' and 'abstract_lexdiv'**

Overall, the results suggest that LLM-generated texts tend to be shorter, more uniform, and optimized for superficial linguistic variation. Human-written texts show greater variability in both structure and content richness.

In addition, the Detectotron 5000 prediction probabilities (pred) were analyzed in figure 4. A histogram of the raw scores shows a heavy right skew, with most predictions falling between 0.0 and 0.2. To better understand the distribution, a log-transformation (log1p) was applied. Despite this, most values remain tightly clustered below 0.1, indicating that the model rarely predicts high probabilities of "fake." This skew presents challenges for modeling tasks, especially regression, and may reflect a conservative bias in the detector toward lower confidence outputs.

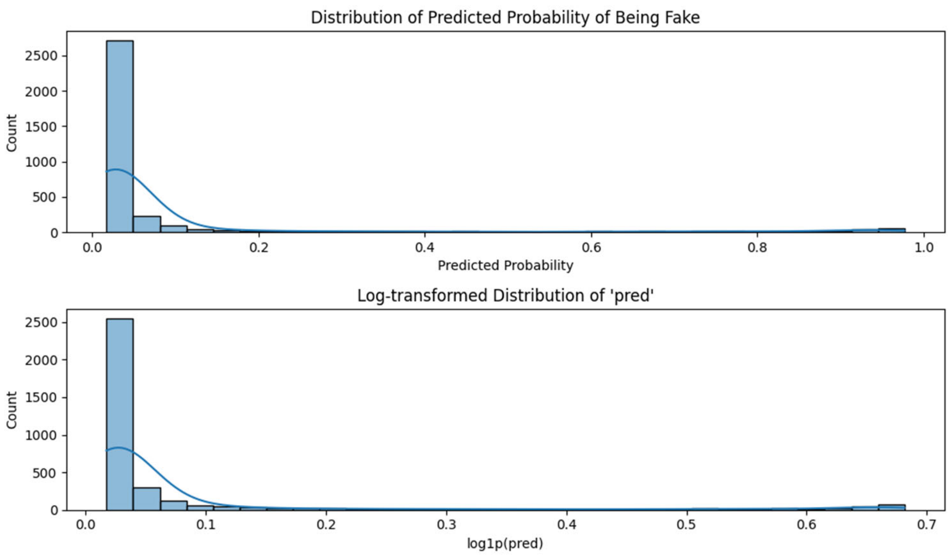


**Figure 4 The histogram of 'Pred' and 'log1p(pred)'**

## Task 1: Binary Classification – Detecting LLM-generated Entries

The goal of this task is to build a binary classifier to predict whether a paper entry was generated by a large language model (LLM) or written by a human. The target variable `generated` is numerically encoded (0 = human, 1 = LLM). Input features are derived by concatenating the `title`, `abstract`, and `TLDR` fields, followed by TF-IDF vectorization (max_features = 5000).

We trained three classifiers—Logistic Regression, Random Forest, and Support Vector Machine (SVM)—with hyperparameter tuning via GridSearchCV and 5-fold cross-validation. Evaluation was performed using accuracy, precision, recall, F1-score (per class), macro F1, and confusion matrices.

Performance Summary (see Table 1): SVM achieved the highest performance with 98.1% accuracy and macro F1 of 0.974, showing balanced precision and recall across both classes. And Logistic Regression followed closely with 97.8% accuracy. In addition, Random Forest attained perfect precision for LLM class (1.000) but suffered from lower recall (0.805), indicating under-detection of generated entries.

| Model | Accuracy | Precision (Human) | Recall (Human) | F1-score (Human) | Precision (LLM) | Recall (LLM) | F1-score (LLM) | Macro F1 |
|---|---|---|---|---|---|---|---|---|
| Logistic Regression | 0.978 | 0.976 | 0.996 | 0.986 | 0.987 | 0.921 | 0.953 | 0.969 |
| Random Forest | 0.954 | 0.943 | 1.000 | 0.971 | 1.000 | 0.805 | 0.892 | 0.931 |
| SVM | 0.981 | 0.983 | 0.992 | 0.988 | 0.975 | 0.945 | 0.960 | 0.974 |

**Table 1 Performance of Logistic Regression, RandomForest and SVM (the actual result is in appendix)**

Confusion Matrix Insights (Figure 1): SVM made the fewest errors overall (13 misclassifications), effectively balancing between overfitting to one class and underfitting the other. Logistic Regression also performed well but showed more confusion in LLM identification. Random Forest, while conservative in LLM predictions, missed more generated samples. These findings suggest that SVM is better at learning complex decision boundaries under sparse feature representations.

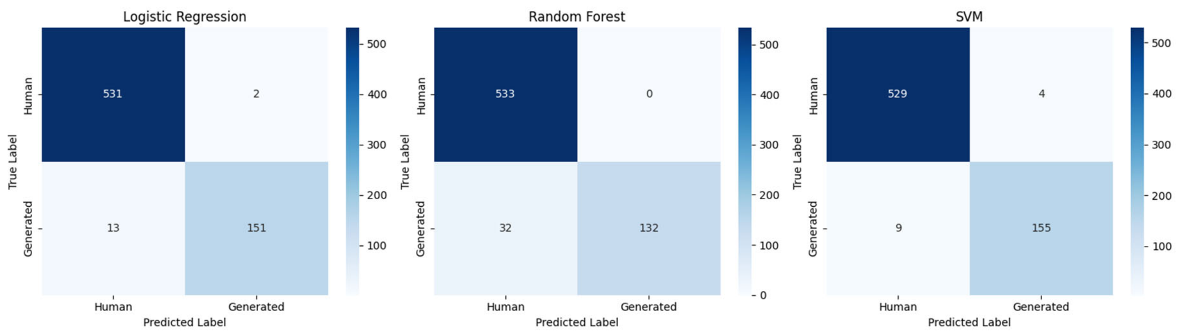


**Figure 5 The confusion matrices for three classification models**

## Task 2: Multiclass Classification – Predicting the Source Model

This task involves building a multiclass classifier to predict the original model (e.g., GPT, Claude, or Human) of a paper entry. The target variable `model` includes multiple LLM classes and one human-written category, with significant class imbalance (Table 1). The input features are the concatenated text fields `title`, `abstract`, and `TLDR`, vectorized using TF-IDF.

| Model | human | deepseek_R1 | gpt_o1 | gemini2.0 | llama3.3_70B | grok3 | claude3.7 |
|---|---|---|---|---|---|---|---|
| Count | 2609 | 200 | 198 | 178 | 100 | 100 | 100 |

**Table 2 The type of model (the actual result is in appendix)**

Model Pipeline & Training: We tested two models—RandomForestClassifier and XGBClassifier—both tuned using GridSearchCV with weighted F1-score (f1_weighted) and 3-fold cross-validation. To address class imbalance, class weights were computed using sklearn.utils.class_weight and applied via sample_weight during training.

Results: The best-performing model was XGBoost, achieving a weighted F1 score of 0.951 on cross-validation and 0.96 accuracy on the test set.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| claude3.7 | 0.91 | 0.67 | 0.77 | 15 |
| deepseek_R1 | 0.94 | 0.92 | 0.93 | 36 |
| gemini2.0 | 0.79 | 1.00 | 0.88 | 30 |
| gpt_o1 | 0.997 | 0.79 | 0.87 | 43 |
| grok3 | 0.89 | 0.84 | 0.86 | 19 |
| human | 0.98 | 0.99 | 0.98 | 533 |
| llama3.3_70B | 0.83 | 0.95 | 0.89 | 21 |
| accuracy | | | 0.96 | 697 |
| macro avg | 0.90 | 0.88 | 0.88 | 697 |
| weighted avg | 0.96 | 0.96 | 0.96 | 697 |

**Table 3 The performance of XGBoost for these models (the actual result is in appendix)**

As shown in Table 2, the classifier performed particularly well on common classes such as human (F1 = 0.98) and deepseek_R1 (F1 = 0.93). However, performance on smaller classes such as claude3.7 was weaker, with a recall of just 0.67, likely due to its limited training samples and semantic similarity to other LLMs such as gemini2.0.

Visual Insights: Figure 5 presents the normalized confusion matrix, showing high confidence on the human class but notable confusion between claude3.7 and gemini2.0.

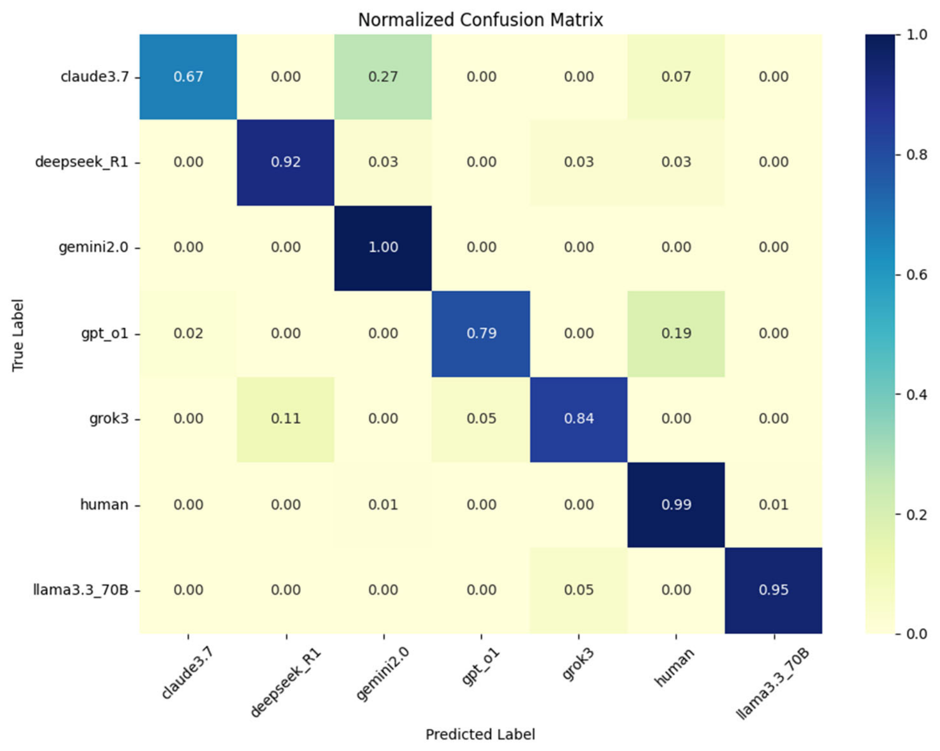


**Figure 6 The normalized confusion matrix of different models**

Figure 6 plots per-class F1 scores, confirming consistent performance across most classes with slight degradation on low-support categories.

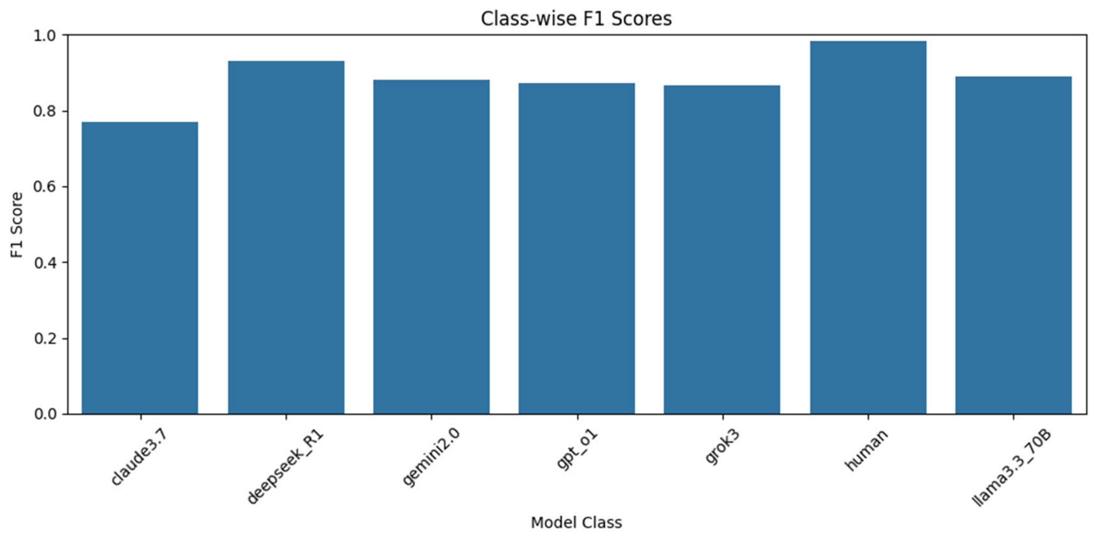


**Figure 7 Class_wise F1 scores for different models**

## Task 3: Regression Analysis – Predicting Confidence Score

To investigate whether the confidence score (pred) could be predicted from available features, we implemented and assessed three regression models with progressively more sophisticated input representations.

Model 1 applied the traditional Bag-of-Words approach, using CountVectorizer to extract sparse token features from title, abstract, and TLDR. These were used to train a RandomForestRegressor with hyperparameter optimization via RandomizedSearchCV. The model achieved an $R^2$ score of 0.1032 and a mean squared error (MSE) of 0.0384, demonstrating modest predictive capability.

Model 2 replaced Bag-of-Words with contextual embeddings generated by Sentence-BERT (multi-qa-mpnet-base-dot-v1). The model utilized only dense vector representations of the text fields and used LightGBM for regression. This approach performed slightly better, achieving the best result among all models with an $R^2$ score of 0.1201 and MSE of 0.0338.

Model 3 extended Model 2 by introducing additional structured features: one-hot encoded categorical fields (generated, model) and basic text statistics (word counts of the title, abstract, and TLDR). Surprisingly, despite having more input features, this model performed significantly worse, yielding an $R^2$ score close to 0.0003, suggesting no useful pattern was learned.

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| MSE | 0.0384 | 0.0338 | 0.0241 |
| $R^2$ score | 0.1041 | 0.1211 | 0.0003 |

**Table 4 The result MSE, R2score of three models (the actual result is in appendix)**

Model 2 was selected as the final regression model due to its best overall performance, even though all three models exhibited low predictive power. The results suggest that the confidence score is either highly subjective, noisy, or driven by information not present in the text or metadata alone.

The scatter plot (Figure 8) above illustrates the regression performance of Model 2, which utilizes Sentence-BERT (multi-qa-mpnet-base-dot-v1) embeddings for title, abstract, and TLDR, combined with a LightGBM regressor. Each dot represents a test sample, plotting its true confidence score (x-axis) against the predicted value (y-axis).

While most predictions are clustered between 0.05 and 0.25, the actual confidence scores are distributed more broadly, with many values extending above 0.6. This mismatch—especially the model's inability to predict higher scores—explains the relatively low $R^2$ score of 0.1201, despite a reasonable mean squared error (MSE) of 0.0338.

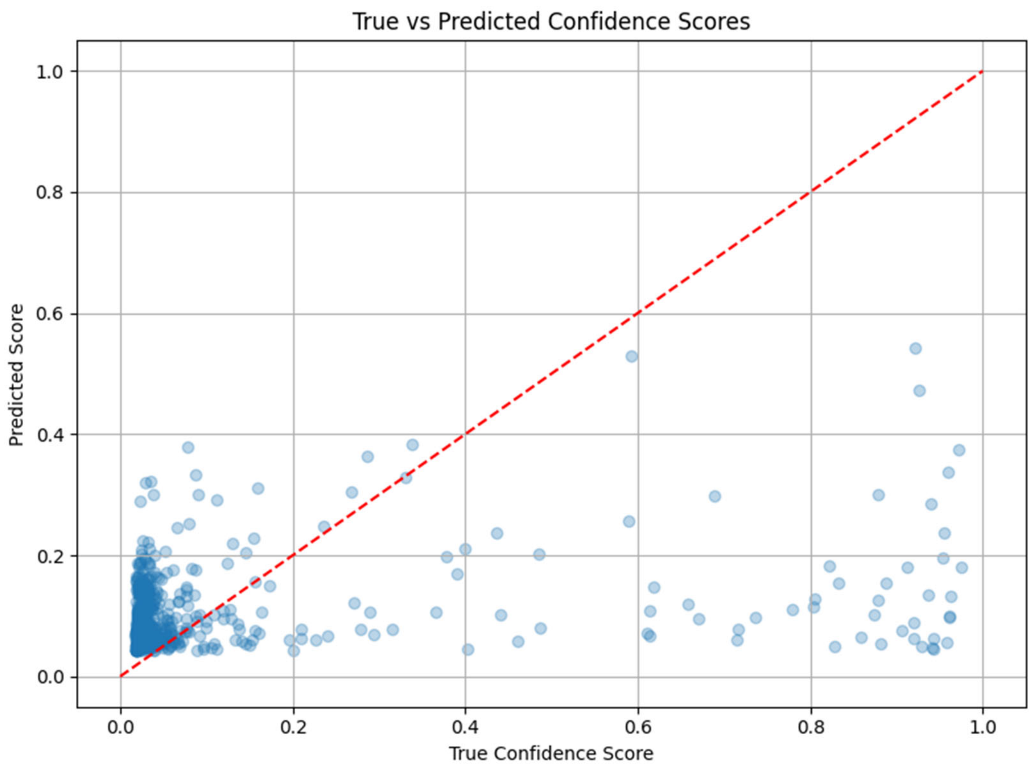


**Figure 8 Predicted vs. True Confidence Scores (Model 2: Sentence-BERT + LightGBM)**

## Conclusion

This project examined the feasibility of using machine learning to detect and analyze LLM-generated academic text. Through exploratory analysis, we found systematic differences between human- and machine-generated content, particularly in length and lexical diversity. Building on this, we defined and evaluated three learning tasks: binary classification, multiclass attribution, and confidence score regression.

Our experiments show that text-based classifiers—especially Support Vector Machines and XGBoost—can reliably distinguish between human and synthetic text and even predict the specific LLM used to generate it. However, confidence score regression proved difficult, likely due to noise in the target variable and the black-box nature of the underlying model.

Overall, the results highlight both the promise and limitations of current machine learning approaches in the context of AI text detection. While classifiers may be useful for real-time filtering and content moderation, more transparent and interpretable scoring systems will be necessary for robust and explainable detection in critical application areas such as academic publishing.

## References

Dojo, D.S. (2023) ‘The Intersection of Human-Computer Interaction and LLMs’. Available at: https://datasciencedojo.com/blog/human-computer-interaction-and-llms/.

Hao, K. (2023) ‘Why detecting AI-generated text is so difficult—and what to do about it’, *MIT Technology Review* [Preprint]. Available at: https://www.technologyreview.com/2023/02/07/1067928/why-detecting-ai-generated-text-is-so-diffficult-and-what-to-do-about-it/.

Kumar, R. and Zhang, L. (2023) ‘LLMEra: Impact of Large Language Models’, *ResearchGate* [Preprint]. Available at: https://www.researchgate.net/publication/381250937_LLMEra_Impact_of_Large_Language_Models.

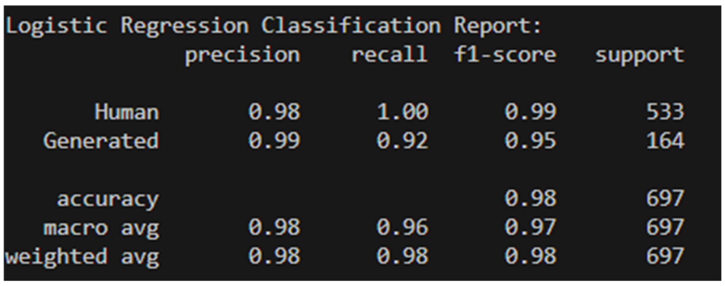
Simonite, T. (2023) ‘Large Language Models Are Reinventing Artificial Intelligence’, *Wired* [Preprint]. Available at: https://www.wired.com/story/large-language-models-artificial-intelligence/.

Society, I.C. (2023) ‘Large Language Models in Modern Business’, *IEEE Tech News* [Preprint]. Available at: https://www.computer.org/publications/tech-news/trends/large-language-models-in-modern-business.

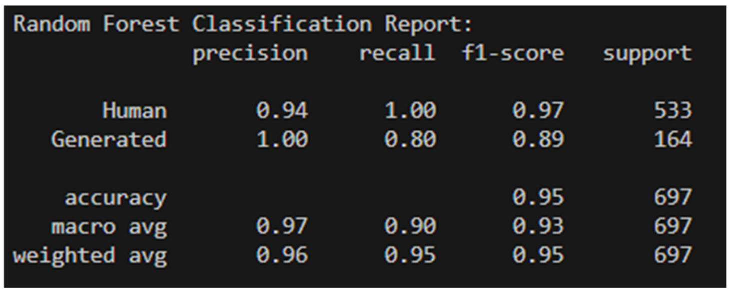
Wikipedia contributors (2024) ‘Large language model’. Available at: https://en.wikipedia.org/wiki/Large_language_model.

## Appendix A

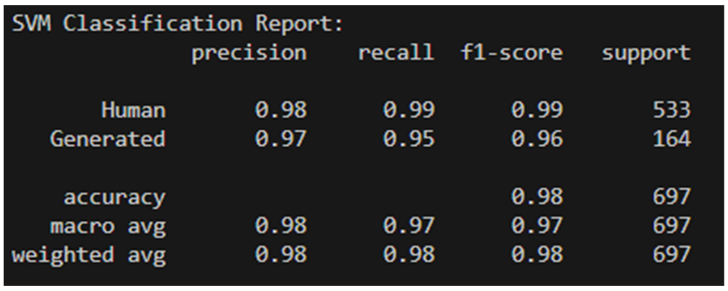
The output responding to Table 1:



```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

       Human       0.98      1.00      0.99       533
   Generated       0.99      0.92      0.95       164

    accuracy                           0.98       697
   macro avg       0.98      0.96      0.97       697
weighted avg       0.98      0.98      0.98       697
```

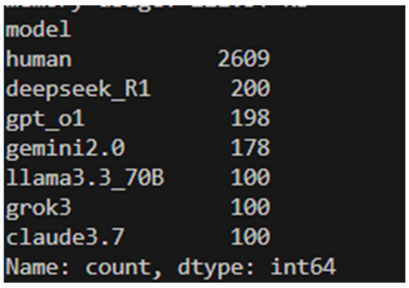


```
Random Forest Classification Report:
              precision    recall  f1-score   support

       Human       0.94      1.00      0.97       533
   Generated       1.00      0.80      0.89       164

    accuracy                           0.95       697
   macro avg       0.97      0.90      0.93       697
weighted avg       0.96      0.95      0.95       697
```



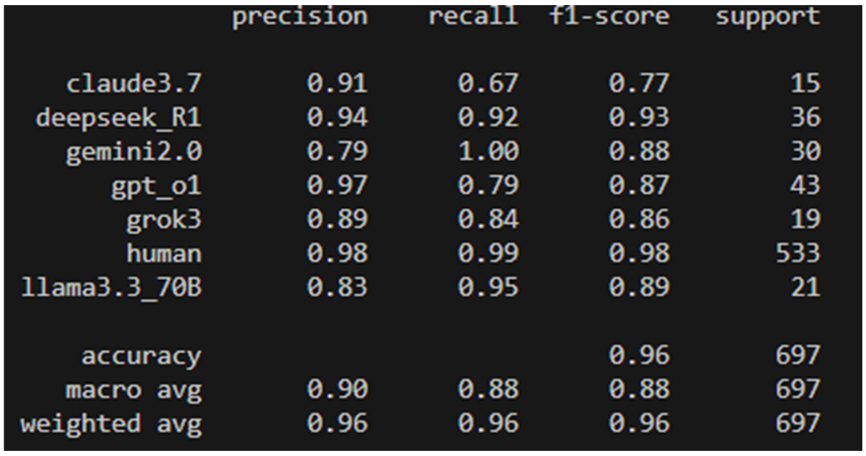
```
SVM Classification Report:
              precision    recall  f1-score   support

       Human       0.98      0.99      0.99       533
   Generated       0.97      0.95      0.96       164

    accuracy                           0.98       697
   macro avg       0.98      0.97      0.97       697
weighted avg       0.98      0.98      0.98       697
```

The output responding to Table 2:



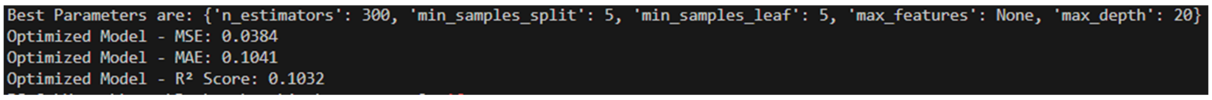
```
model
human           2609
deepseek_R1      200
gpt_o1           198
gemini2.0        178
llama3.3_70B     100
grok3            100
claude3.7        100
Name: count, dtype: int64
```

The output responding to Table 3:

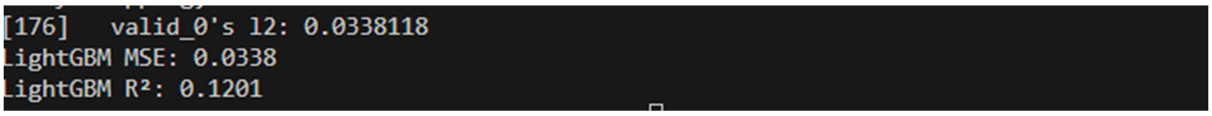


```
              precision    recall  f1-score   support

   claude3.7       0.91      0.67      0.77        15
 deepseek_R1       0.94      0.92      0.93        36
   gemini2.0       0.79      1.00      0.88        30
      gpt_o1       0.97      0.79      0.87        43
       grok3       0.89      0.84      0.86        19
       human       0.98      0.99      0.98       533
llama3.3_70B       0.83      0.95      0.89        21

    accuracy                           0.96       697
   macro avg       0.90      0.88      0.88       697
weighted avg       0.96      0.96      0.96       697
```

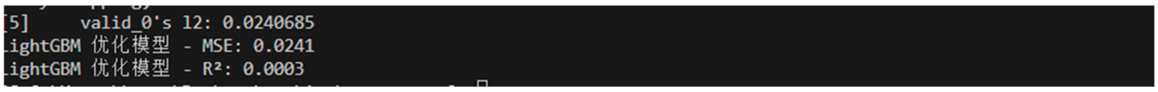
The output responding to Table 4:



```
Best Parameters are: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf': 5, 'max_features': None, 'max_depth': 20}
Optimized Model - MSE: 0.0384
Optimized Model - MAE: 0.1041
Optimized Model - R² Score: 0.1032
```



```
[176]   valid_0's l2: 0.0338118
LightGBM MSE: 0.0338
LightGBM R²: 0.1201
```



```
[5]     valid_0's l2: 0.0240685
LightGBM 优化模型 - MSE: 0.0241
LightGBM 优化模型 - R²: 0.0003
```

The other scatter plot in task 3:

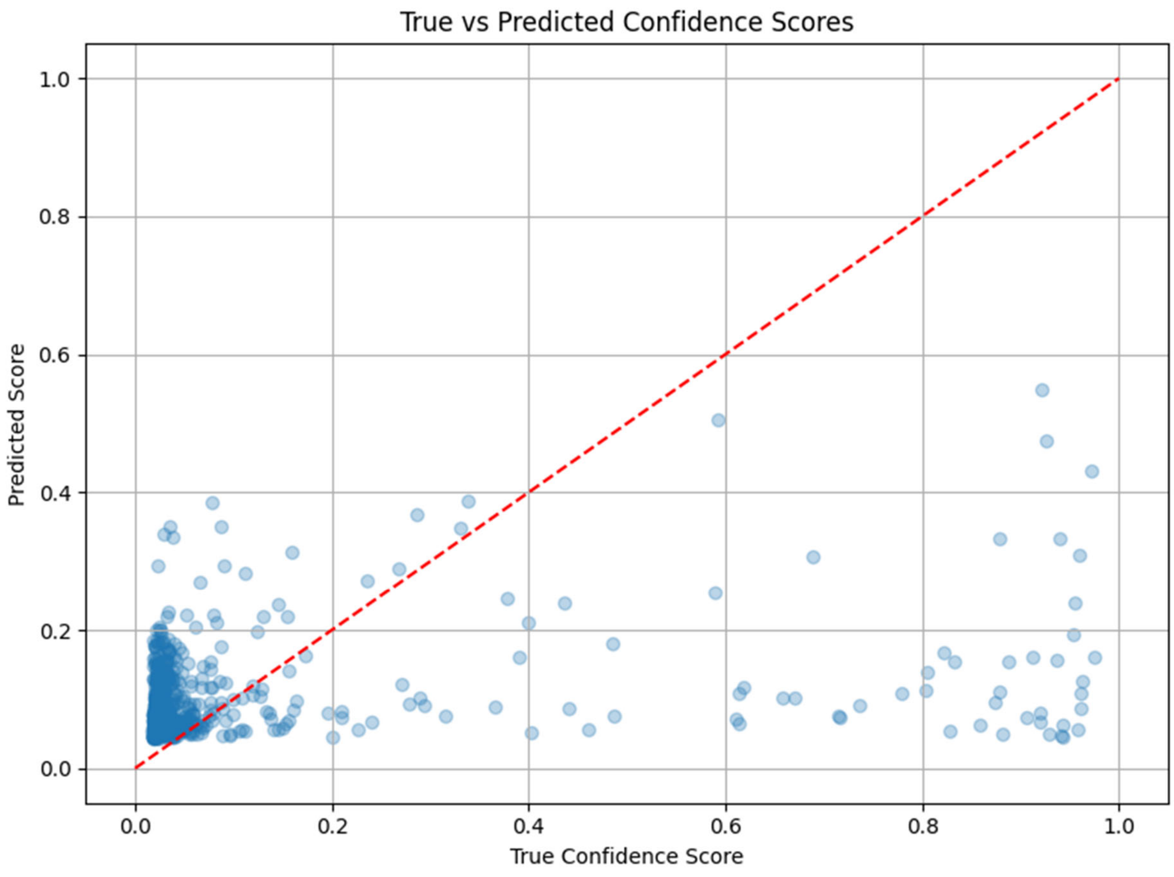


**Figure 9 Predicted vs. True Confidence Scores (Model 1: CountVectorizer + RandomForestRegressor )**

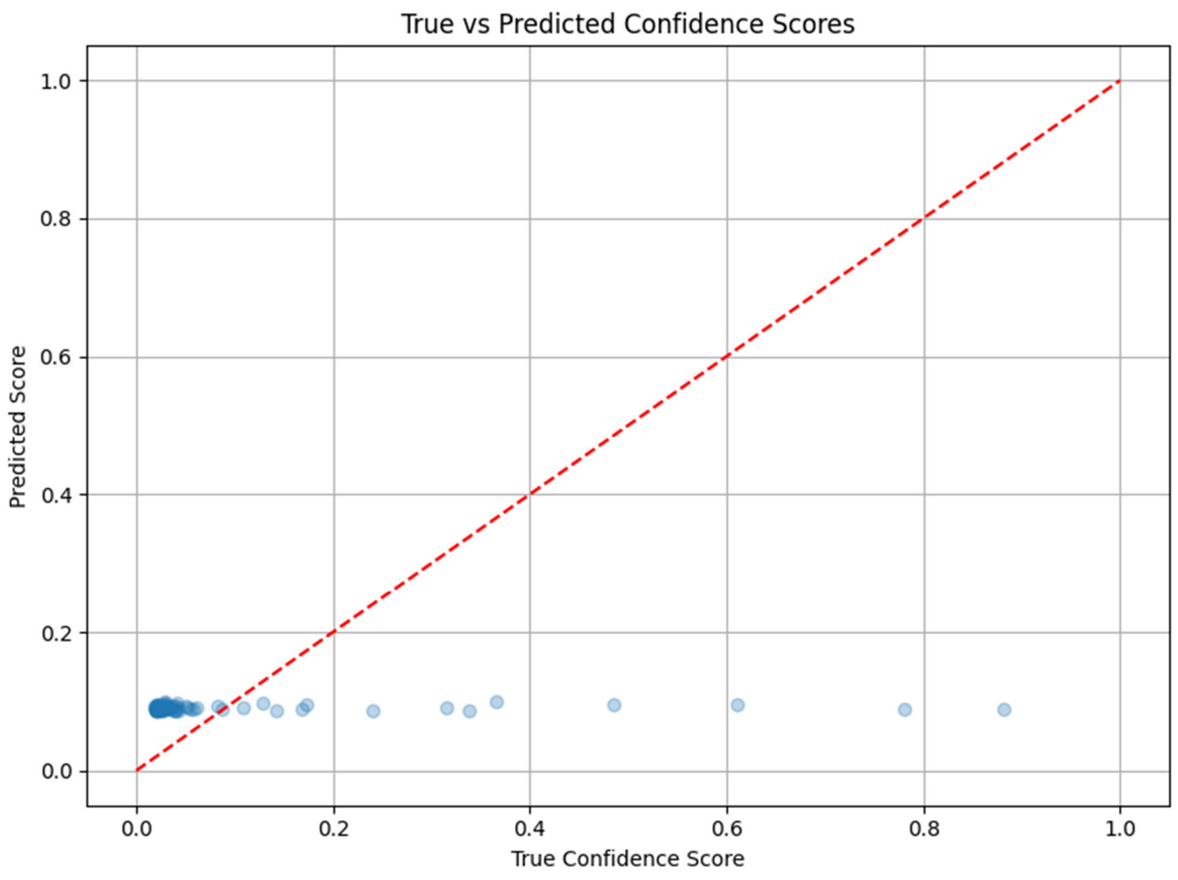


**Figure 10 Predicted vs. True Confidence Scores (Model 3: One-hot encode + LightGBM)**

## Appendix B: Code for Dataset Summary & Visualisations

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df =
 pd.read_csv(r"C:\Users\Lance\Desktop\machine\assessment2\cw2_dataset_read_only.cs
 v")

# Display options
pd.set_option('display.width', 200)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 100)
print(df.head)

# Define helper functions to calculate word count and lexical diversity
def word_count(text):
    return len(str(text).split())

def lexical_diversity(text):
    tokens = str(text).split()
    if len(tokens) == 0:
        return 0
    return len(set(tokens)) / len(tokens)

# Compute word counts and lexical diversity for title, abstract, and TLDR
df['title_len'] = df['title'].apply(word_count)
df['abstract_len'] = df['abstract'].apply(word_count)
df['tldr_len'] = df['TLDR'].apply(word_count)
df['title_lexdiv'] = df['title'].apply(lexical_diversity)
df['abstract_lexdiv'] = df['abstract'].apply(lexical_diversity)
df['tldr_lexdiv'] = df['TLDR'].apply(lexical_diversity)

# Map boolean column to categorical source label
df['source'] = df['generated'].map({False: 'Human', True: 'LLM'})

# ---------------- Summary Statistics ----------------
# Compute average word counts and lexical diversity by source
summary = df.groupby('source')[['title_len', 'abstract_len', 'tldr_len',
 'title_lexdiv', 'abstract_lexdiv', 'tldr_lexdiv']].mean()
print("Average word counts and lexical diversity:\n", summary)
```

```
# Count the number of generated vs. human-written papers
generated_counts = df['generated'].value_counts()
print("\nCount of generated vs non-generated papers:\n", generated_counts)

# Manually calculate the accuracy of Detectotron 5000
df['generated_binary'] = df['generated'].astype(int)
df['pred_label'] = (df['pred'] >= 0.5).astype(int)
correct_predictions = (df['generated_binary'] == df['pred_label']).sum()
total_samples = len(df)
manual_accuracy = correct_predictions / total_samples
print(f"\nAccuracy of Detectotron 5000: {manual_accuracy:.2%}")

# ---------------- Visualisations ----------------
# Boxplot: Abstract length by source
plt.figure(figsize=(10, 8))
plt.subplot(2, 2, 1)
sns.boxplot(x='source', y='abstract_len', data=df)
plt.title('Distribution of Abstract Length by Source')
plt.ylabel('Word Count')
plt.xlabel('Source')
plt.tight_layout()

# Boxplot: Title length by source
plt.subplot(2, 2, 2)
sns.boxplot(x='source', y='title_len', data=df)
plt.title('Distribution of Title Length by Source')
plt.ylabel('Word Count')
plt.xlabel('Source')
plt.tight_layout()

# Boxplot: TLDR length by source
plt.subplot(2, 2, 3)
sns.boxplot(x='source', y='tldr_len', data=df)
plt.title('Distribution of TLDR Length by Source')
plt.ylabel('Word Count')
plt.xlabel('Source')
plt.tight_layout()

# Boxplot: Abstract lexical diversity by source
plt.subplot(2, 2, 4)
sns.boxplot(x='source', y='abstract_lexdiv', data=df)
plt.title('Distribution of Abstract Lexical Diversity by Source')
plt.ylabel('Lexical Diversity (Unique Words / Total Words)')
plt.xlabel('Source')
plt.tight_layout()
plt.show()

# Histogram: Predicted probability of being fake
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
sns.histplot(df['pred'], bins=30, kde=True)
plt.title("Distribution of Predicted Probability of Being Fake")
plt.xlabel("Predicted Probability")
plt.ylabel("Count")
plt.tight_layout()

# Histogram: Log-transformed predicted score
plt.subplot(2, 1, 2)
df['pred_log'] = np.log1p(df['pred'])
sns.histplot(df['pred_log'], bins=30, kde=True)
plt.title("Log-transformed Distribution of 'pred'")
plt.xlabel("log1p(pred)")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```

## Appendix C: Code for Task 1 – Binary Classification (Generated vs. Human)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Load the dataset
df =
pd.read_csv(r"C:\Users\Lance\Desktop\machine\assessment2\cw2_dataset_read_only.csv
")

# Combine textual fields into a single feature
df['full_text'] = df['title'] + ' ' + df['abstract'] + ' ' + df['TLDR']
X = df['full_text']
y = df['generated'].astype(int)  # 0 = Human, 1 = LLM

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# TF-IDF vectorisation
vectorizer = TfidfVectorizer(max_features=5000)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Logistic Regression with GridSearchCV
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['liblinear', 'lbfgs']
}
lr = LogisticRegression(max_iter=1000)
grid_lr = GridSearchCV(lr, param_grid_lr, cv=5, scoring='f1', n_jobs=-1)
grid_lr.fit(X_train_vec, y_train)
best_lr = grid_lr.best_estimator_
lr_preds = best_lr.predict(X_test_vec)
print("Best Logistic Regression Params:", grid_lr.best_params_)
```

```
# Random Forest with GridSearchCV
param_grid_rf = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5]
}
rf = RandomForestClassifier(random_state=42)
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1', n_jobs=-1)
grid_rf.fit(X_train_vec, y_train)
best_rf = grid_rf.best_estimator_
rf_preds = best_rf.predict(X_test_vec)
print("Best Random Forest Params:", grid_rf.best_params_)

# Support Vector Machine with GridSearchCV
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf']
}
svm = SVC()
grid_svm = GridSearchCV(svm, param_grid_svm, cv=5, scoring='f1', n_jobs=-1)
grid_svm.fit(X_train_vec, y_train)
best_svm = grid_svm.best_estimator_
svm_preds = best_svm.predict(X_test_vec)
print("Best SVM Params:", grid_svm.best_params_)

# Evaluation function
def evaluate_model(name, y_true, y_pred):
    print(f"\n{name} Classification Report:")
    print(classification_report(y_true, y_pred, target_names=["Human",
"Generated"]))

# Evaluate all models
evaluate_model("Logistic Regression", y_test, lr_preds)
evaluate_model("Random Forest", y_test, rf_preds)
evaluate_model("SVM", y_test, svm_preds)

# Confusion Matrix Visualisation
model_preds = {
    "Logistic Regression": lr_preds,
    "Random Forest": rf_preds,
    "SVM": svm_preds
}

fig, axes = plt.subplots(1, 3, figsize=(18, 5))
```

```
for ax, (model_name, y_pred) in zip(axes, model_preds.items()):
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Human", "Generated"],
                yticklabels=["Human", "Generated"],
                ax=ax)
    ax.set_title(f"{model_name}")
    ax.set_xlabel("Predicted Label")
    ax.set_ylabel("True Label")

plt.suptitle("Confusion Matrices for Three Classification Models", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig("task1_confusion_matrices.png", dpi=300, bbox_inches='tight')
plt.show()
```

## Appendix D: Code for Task 2 – Multiclass Classification (Model Type)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.utils.class_weight import compute_class_weight

# 1. Load the dataset and concatenate text fields
df = pd.read_csv("cw2_dataset_read_only.csv")
df["text"] = df["title"].fillna("") + " " + df["abstract"].fillna("") + " " +
df["TLDR"].fillna("")

X = df["text"]
y = df["model"]

# 2. Encode class labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 3. Split into training and test sets
X_train, X_test, y_train_enc, y_test_enc = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

# 4. Compute class weights for sample balancing
classes = np.unique(y_train_enc)
class_weights = compute_class_weight(class_weight='balanced', classes=classes,
y=y_train_enc)
weight_dict = dict(zip(classes, class_weights))
sample_weights = pd.Series(y_train_enc).map(weight_dict).values  # convert to
numpy array

# 5. Define pipeline with TF-IDF and classifiers
tfidf = TfidfVectorizer(max_features=10000)
pipeline = Pipeline([
    ("tfidf", tfidf),
    ("clf", RandomForestClassifier())
])

# Grid search parameter space for RF and XGBoost
param_grid = [
    {
        "clf": [RandomForestClassifier(random_state=42)],
        "clf__n_estimators": [100, 200],
        "clf__max_depth": [10, None],
    },
    {
        "clf": [XGBClassifier(use_label_encoder=False, eval_metric="mlogloss",
verbosity=0)],
        "clf__n_estimators": [100, 200],
        "clf__max_depth": [4, 6],
        "clf__learning_rate": [0.01, 0.1],
    }
]

# 6. Perform grid search (single-threaded to avoid parallel errors)
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=3,
    scoring="f1_weighted",
    verbose=2,
    n_jobs=1
)

# 7. Fit the model with sample weights
grid_search.fit(X_train, y_train_enc, clf__sample_weight=sample_weights)

# 8. Display best model and CV score
best_model = grid_search.best_estimator_
print("Best parameters:", grid_search.best_params_)
print("Best F1 (CV):", grid_search.best_score_)

# 9. Predict on test set and decode labels
y_pred_enc = best_model.predict(X_test)
y_pred = label_encoder.inverse_transform(y_pred_enc)
y_test = label_encoder.inverse_transform(y_test_enc)

# 10. Classification report
print(classification_report(y_test, y_pred))
```

```python
# 11. Normalized confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=label_encoder.classes_)
cm_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(10, 8))
sns.heatmap(cm_norm, annot=True, fmt=".2f", cmap="YlGnBu",
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Normalized Confusion Matrix")
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# 12. Class-wise F1 score bar chart
report = classification_report(y_test, y_pred, output_dict=True)
f1_scores = {label: report[label]["f1-score"] for label in label_encoder.classes_}

plt.figure(figsize=(10, 5))
sns.barplot(x=list(f1_scores.keys()), y=list(f1_scores.values()))
plt.ylabel("F1 Score")
plt.xlabel("Model Class")
plt.title("Class-wise F1 Scores")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

## Appendix C: Code for Task 3 – Regression (Confidence Score)

Model 1 – CountVectorizer + Random Forest Regressor

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from scipy.sparse import hstack
import matplotlib.pyplot as plt

# 1. Load dataset
df =
 pd.read_csv(r"C:\Users\Lance\Desktop\machine\assessment2\cw2_dataset_read_only.cs
 v")

# 2. Separate features and target variable
X = df.drop(columns=['pred'])         # Features
Pred = df['pred']                     # Target: predicted confidence score

# Split into training and test sets
df_train, df_test, Pred_train, Pred_test = train_test_split(
    X, Pred, test_size=0.2, random_state=22
)

# 3. Define stopwords
stop_words = text.ENGLISH_STOP_WORDS.union(["br"])
stop_words = list(stop_words)

# 4. Vectorize abstract, title, and TLDR using CountVectorizer
vectorizer_abs = CountVectorizer(strip_accents="unicode", stop_words=stop_words)
vectorizer_ti = CountVectorizer(strip_accents="unicode", stop_words=stop_words)
vectorizer_tldr = CountVectorizer(strip_accents="unicode", stop_words=stop_words)

df_traina = vectorizer_abs.fit_transform(df_train["abstract"])
df_testa = vectorizer_abs.transform(df_test["abstract"])

df_trainti = vectorizer_ti.fit_transform(df_train["title"])
df_testti = vectorizer_ti.transform(df_test["title"])

df_traintl = vectorizer_tldr.fit_transform(df_train["TLDR"])
```

```
df_testtl = vectorizer_tldr.transform(df_test["TLDR"])

# 5. Combine all text feature vectors into sparse matrices
X_train = hstack([df_traina, df_trainti, df_traintl])
X_test = hstack([df_testa, df_testti, df_testtl])

# 6. Hyperparameter tuning using RandomizedSearchCV for RandomForestRegressor
rf = RandomForestRegressor(random_state=42)
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['sqrt', 'log2', None]
}

random_search = RandomizedSearchCV(
    rf,
    param_distributions=param_dist,
    n_iter=20,
    cv=5,
    verbose=2,
    n_jobs=-1,
    scoring='r2'
)

random_search.fit(X_train, Pred_train)

# 7. Train final model using best parameters
best_params = random_search.best_params_
print(f"Best Parameters are: {best_params}")

final_model = RandomForestRegressor(**best_params, random_state=42)
final_model.fit(X_train, Pred_train)

# 8. Predict and evaluate on the test set
Pred_test_pred = final_model.predict(X_test)

mse = mean_squared_error(Pred_test, Pred_test_pred)
r2 = r2_score(Pred_test, Pred_test_pred)
mae = mean_absolute_error(Pred_test, Pred_test_pred)

print(f'Optimized Model - MSE: {mse:.4f}')
print(f'Optimized Model - MAE: {mae:.4f}')
print(f'Optimized Model - R² Score: {r2:.4f}')

# 9. Plot predicted vs. true scores
plt.figure(figsize=(8, 6))
plt.scatter(Pred_test, Pred_test_pred, alpha=0.3)
plt.plot([0, 1], [0, 1], '--', color='red')  # Reference line
plt.xlabel("True Confidence Score")
plt.ylabel("Predicted Score")
plt.title("True vs Predicted Confidence Scores")
plt.grid(True)
plt.tight_layout()
plt.show()
```

## Model 2 – Sentence-BERT Embeddings + LightGBM + Structured Features

```
import pandas as pd
import numpy as np
from sentence_transformers import SentenceTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from lightgbm import Dataset, train, callback
import matplotlib.pyplot as plt

# Step 1: Load the dataset
df = pd.read_csv(r"C:\Users\Lance\Desktop\machine\assessment2\output.csv")

# Step 2: Generate BERT embeddings for text features
bert_model = SentenceTransformer("multi-qa-mpnet-base-dot-v1")

df["title_embedding"] = df["title"].apply(lambda x: bert_model.encode(str(x)))
df["abstract_embedding"] = df["abstract"].apply(lambda x:
 bert_model.encode(str(x)))
df["tldr_embedding"] = df["TLDR"].apply(lambda x: bert_model.encode(str(x)))

X_text = np.hstack([
    np.vstack(df["title_embedding"]),
    np.vstack(df["abstract_embedding"]),
    np.vstack(df["tldr_embedding"])
])

# Step 3: Add simple numeric text features
df["title_length"] = df["title"].apply(lambda x: len(str(x).split()))
```

```
df["abstract_length"] = df["abstract"].apply(lambda x: len(str(x).split()))
df["tldr_length"] = df["TLDR"].apply(lambda x: len(str(x).split()))
X_numeric = df[["title_length", "abstract_length", "tldr_length"]].values

# Step 4: One-hot encode categorical variables
encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
X_categorical = encoder.fit_transform(df[["generated", "model"]])

# Step 5: Combine all features
X = np.hstack([X_text, X_numeric, X_categorical])
y = df["pred"].values

# Step 6: Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=42)

# Step 7: Construct LightGBM datasets
lgb_train = Dataset(X_train, label=y_train)
lgb_eval = Dataset(X_test, label=y_test, reference=lgb_train)

# Step 8: Define LightGBM parameters
params = {
    'objective': 'regression',
    'metric': 'mse',
    'boosting_type': 'gbdt',
    'learning_rate': 0.01,
    'num_leaves': 50,
    'max_depth': 10,
    'min_child_samples': 20,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 0.1,
    'reg_lambda': 0.1,
    'verbose': 2
}

# Step 9: Train the model with early stopping
model = train(
    params,
    lgb_train,
    valid_sets=[lgb_eval],
    num_boost_round=200,
    callbacks=[callback.early_stopping(stopping_rounds=10, verbose=True)]
)
```

```
# Step 10: Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"LightGBM - MSE: {mse:.4f}")
print(f"LightGBM - R²: {r2:.4f}")

# Step 11: Plot predictions vs true scores
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([0, 1], [0, 1], '--', color='red')
plt.xlabel("True Confidence Score")
plt.ylabel("Predicted Score")
plt.title("True vs Predicted Confidence Scores")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Model 3 – Sentence-BERT Embeddings + LightGBM

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from scipy.sparse import hstack
import matplotlib.pyplot as plt

# 1. Load dataset
df =
pd.read_csv(r"C:\Users\Lance\Desktop\machine\assessment2\cw2_dataset_read_only.csv
")

# 2. Separate features and target variable
X = df.drop(columns=['pred'])         # Features
Pred = df['pred']                     # Target: predicted confidence score

# Split into training and test sets
df_train, df_test, Pred_train, Pred_test = train_test_split(X, Pred, test_size=0.2,
random_state=22)

# 3. Define stopwords
stop_words = text.ENGLISH_STOP_WORDS.union(["br"])
stop_words = list(stop_words)

# 4. Vectorize abstract, title, and TLDR using CountVectorizer
vectorizer_abs = CountVectorizer(strip_accents="unicode", stop_words=stop_words)
vectorizer_ti = CountVectorizer(strip_accents="unicode", stop_words=stop_words)
vectorizer_tldr = CountVectorizer(strip_accents="unicode", stop_words=stop_words)

df_traina = vectorizer_abs.fit_transform(df_train["abstract"])
df_testa = vectorizer_abs.transform(df_test["abstract"])
df_trainti = vectorizer_ti.fit_transform(df_train["title"])
df_testti = vectorizer_ti.transform(df_test["title"])
df_traintl = vectorizer_tldr.fit_transform(df_train["TLDR"])
df_testtl = vectorizer_tldr.transform(df_test["TLDR"])

# 5. Combine all text feature vectors
X_train = hstack([df_traina, df_trainti, df_traintl])
X_test = hstack([df_testa, df_testti, df_testtl])

# 6. Hyperparameter tuning using RandomizedSearchCV for RandomForestRegressor
rf = RandomForestRegressor(random_state=42)
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['sqrt', 'log2', None]
}
random_search = RandomizedSearchCV(rf, param_distributions=param_dist, n_iter=20,
cv=5, verbose=2, n_jobs=-1, scoring='r2')
random_search.fit(X_train, Pred_train)

# 7. Train final model using best parameters
best_params = random_search.best_params_
final_model = RandomForestRegressor(**best_params, random_state=42)
final_model.fit(X_train, Pred_train)

# 8. Predict and evaluate on the test set
Pred_test_pred = final_model.predict(X_test)
mse = mean_squared_error(Pred_test, Pred_test_pred)
r2 = r2_score(Pred_test, Pred_test_pred)
```

```
mae = mean_absolute_error(Pred_test, Pred_test_pred)
print(f'MSE: {mse:.4f}, MAE: {mae:.4f}, R²: {r2:.4f}')

# 9. Plot predicted vs. true scores
plt.figure(figsize=(8, 6))
plt.scatter(Pred_test, Pred_test_pred, alpha=0.3)
plt.plot([0, 1], [0, 1], '--', color='red')  # Reference line
plt.xlabel("True Confidence Score")
plt.ylabel("Predicted Score")
plt.title("True vs Predicted Confidence Scores")
plt.grid(True)
plt.tight_layout()
plt.show()
C.2 Model 2 – Sentence-BERT + LightGBM + Structured Features
(Insert Model 2 cleaned code here...)
C.3 Model 3 – Sentence-BERT + LightGBM
# Task 3 – Model 3: Sentence-BERT + One-Hot Features + LightGBM

import pandas as pd
import numpy as np
import lightgbm as lgb
from sentence_transformers import SentenceTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from lightgbm import callback
import matplotlib.pyplot as plt

# Step 1: Load dataset
df =
pd.read_csv(r"C:\Users\Lance\Desktop\machine\assessment2\cw2_dataset_read_only.csv
")

# Step 2: Generate BERT embeddings for text fields
bert_model = SentenceTransformer('all-MiniLM-L6-v2')

df["title_embedding"] = df["title"].apply(lambda x: bert_model.encode(x))
df["abstract_embedding"] = df["abstract"].apply(lambda x: bert_model.encode(x))
df["tldr_embedding"] = df["TLDR"].apply(lambda x: bert_model.encode(x))

# Combine embeddings by summing (not concatenating)
X_text = np.vstack(df["title_embedding"]) + np.vstack(df["abstract_embedding"]) +
np.vstack(df["tldr_embedding"])
```

```
# Step 3: Encode categorical variables using one-hot encoding
categorical_features = ["generated", "model"]
encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
X_categorical = encoder.fit_transform(df[categorical_features])

# Step 4: Combine all features
X = np.hstack([X_text, X_categorical])
y = df["pred"].values

# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 6: Prepare LightGBM datasets
lgb_train = lgb.Dataset(X_train, label=y_train)
lgb_eval = lgb.Dataset(X_test, label=y_test, reference=lgb_train)

# Step 7: Define model parameters
params = {
    'objective': 'regression',
    'metric': 'mse',
    'boosting_type': 'gbdt',
    'learning_rate': 0.01,
    'num_leaves': 50,
    'max_depth': 10,
    'min_child_samples': 20,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 0.1,
    'reg_lambda': 0.1,
    'verbose': 2
}

# Step 8: Train the LightGBM model with early stopping
model = lgb.train(
    params,
    lgb_train,
    valid_sets=[lgb_eval],
    num_boost_round=200,
    callbacks=[callback.early_stopping(stopping_rounds=10, verbose=True)]
)

# Step 9: Predict and evaluate
y_pred = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"LightGBM MSE: {mse:.4f}")
print(f"LightGBM R²: {r2:.4f}")

# Step 10: Plot predicted vs actual scores
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([0, 1], [0, 1], '--', color='red')  # Reference line
plt.xlabel("True Confidence Score")
plt.ylabel("Predicted Score")
plt.title("True vs Predicted Confidence Scores")
plt.grid(True)
plt.tight_layout()
plt.show()
```