CST3133 Advanced Topics in Data Science and Artificial Intelligence Final Report

Machine Learning Workflow Model Development and Ethical Considerations

Natural Language Processing and Deep Learning for Text Analysis

Created By Students:	Student ID:
Arbaaz Ali Hussain	M00872279
Nathan Amani Kagoro	M00910099
Cristian Tcaci	M00674787
ChingHsuan Tsai	M01001570

Module Teachers: Dr Halil Yetgin, Dr Peter Passmore, and Dr Florian Kammueller

Module Leader: Dr Halil Yetgin

Submission date: 16/04/25, 16:00 pm

Faculty of Science and Technology

Department of Computer Science

Middlesex University London

Table of Contents

Machine Learning Workflow and Ethics

Dataset Selection and Problem Definition

In this section, we will use a dataset to expand our knowledge of machine learning algorithms. We chose to explore a "student performance dataset" sourced from Kaggle. It contains various factors that may influence exam scores. This dataset was chosen for its variety of data types and large set of unique instances, coming in at over 11,100 unique instances.

The dataset includes both numeric and categorical variables, as well as missing values and inconsistent placeholders (e.g., "Not Available", "N/A", "NULL"). Our primary goal is to:

- Clean and standardise the data (fix missing values, unify categories, handle outliers, etc.)
- Perform a comprehensive EDA (univariate, bivariate, and correlation analysis)
- Generate preliminary insights about relationships in the data
- Build and evaluate machine learning models to predict student exam scores

This is a regression problem, as we aim to predict a continuous target variable (Exam_Score) based on various predictors. We also explore converting it to a classification problem by grouping scores into ranges.

The dataset contains the following columns:

Numeric Categories

- Hours_Studied
- Attendance
- Previous_Scores
- Exam_Score
- Sleep_Hours
- Tutoring_Sessions
- Physical_Activity

Ordinal Categories (Low, Medium, High)

- Parental_Involvement
- Access_to_Resources
- Teacher_Quality
- Extracurricular_Activities
- Motivation_Level
- Family_Income

Other Categorical Variables:

- Internet_Access (Yes/No)

- Gender (Male/Female)
- School_Type (Public/Private)
- Learning_Disabilities (Yes/No)
- Peer_Influence (Positive/Neutral/Negative)
- Distance_from_Home (Near/Moderate/Far)
- Parental_Education_Level (High School/College/Post Graduate)

Data Preprocessing

We loaded the dataset from a CSV file using Pandas. We specified a list of placeholders (["--", "N/A", "NULL", "Not Available", "Unknown", ...]) so that they would be read as NaN. Then we performed initial inspections.

We used Pandas' read_csv function with a comprehensive list of missing value indicators to ensure consistent handling of various placeholders in the dataset.

Observations

- The dataset had missing values in many columns (both numeric and categorical)
- Some numeric columns (e.g., Hours_Studied, Previous_Scores) contained textual values (like "sixty-five")
- Ordinal categories ("Low", "Medium", "High") appeared in inconsistent forms (like "low", "high")

- ● Certain columns (like Attendance) had partial data

Removing Duplicates

We removed duplicate rows to ensure each row is unique, then reset the index.

Converting Written-Out Numbers to Digits

We noticed entries like "sixty-five". We wrote a helper function that:

1. Splits words like "twenty-three" into numeric form
2. Returns NaN if unrecognised

We applied this to columns we expected to be numeric (like Hours_Studied, Sleep_Hours, Previous_Scores, etc.). This ensures we have actual numeric data rather than textual strings.

Forcing Numeric Columns

After dealing with textual numbers, we used `pd.to_numeric(errors='coerce')` to turn any leftover strings into NaN.

We identified columns such as:

- Numeric: Hours_Studied, Attendance, Sleep_Hours, Previous_Scores, Exam_Score, Tutoring_Sessions,
  Physical_Activity, Family_Income
- Any unconvertible entries became NaN

Handling Missing Values

For **Numeric Columns**, we imputed missing values with the median of each column. For **Categorical Columns**, we imputed missing values with the mode (most frequent category). This ensures that no columns remain with NaN values.

Normalising Inconsistent Strings

Some columns had variations like "Male"/"male"/"MALE". We standardised them by converting everything to lowercase, trimming spaces, and mapping them to consistent forms:

- "low" → "Low", "medium" → "Medium", "high" → "High"
- "male" → "Male", "female" → "Female"
- "yes" → "Yes", "no" → "No"

Mapping Categorical Variables

We applied one-hot encoding to categorical columns to prepare them for machine learning models:

Outlier Clipping & Scaling

For columns like Hours_Studied (which might have extreme values):

- We clipped outliers at the 1st and 99th percentiles
- We applied standardisation to centre the data with mean=0 and standard deviation=1

Final Check

We verified that all preprocessing steps were successful by:

- Checking that no missing values remained using `df.isnull().sum()`
- Examining the updated descriptive statistics with `df.describe()`
- Confirming that each column was in the expected data type with `df.info()`

Exploratory Data Analysis

Univariate Analysis

Histograms / KDE for numeric columns: We plotted the distribution of columns like Hours_Studied, Attendance, Sleep_Hours, etc. We found that some columns were slightly skewed (e.g., Hours_Studied might have a tail above 30). Others were fairly normally distributed.

Boxplots for numeric columns: This helped us spot outliers. For example, Exam_Score ranged from 50 to 100, with few outliers beyond typical bounds.

Countplots for categorical columns: We checked the frequency distribution of categories like Teacher_Quality (Low, Medium, High). We observed that "Medium" was often the most common category in columns like Parental_Involvement.

Bivariate Analysis

Pairplot among numeric columns: We used `sns.pairplot` to see pairwise scatterplots. This highlighted linear relationships. We observed that Hours_Studied showed a moderate positive correlation with Exam_Score.

Scatterplots vs. Exam_Score: We individually plotted each numeric column (like Attendance) vs. Exam_Score. We found a moderate correlation for Previous_Scores vs. Exam_Score.

Boxplots / Violin Plots for categorical columns vs. Exam_Score: For example, we examined Teacher_Quality
(Low/Medium/High) vs. Exam_Score. We found that students in "High" teacher-quality classes had a higher median exam score than those in "Low" or "Medium" classes.

Multivariate Analysis: Correlation Heatmap

We created a correlation heatmap for numeric features using `sns.heatmap(df.corr(), annot=True)`. Some highlights:

- Exam_Score showed a positive correlation with Previous_Scores and Hours_Studied
- Attendance was modestly correlated with Exam_Score
- Some features were nearly uncorrelated with Exam_Score (e.g., Distance_from_Home)

Preliminary Findings & Insights

Strongest Predictors: Previous_Scores had the highest correlation with Exam_Score. Hours_Studied also showed a moderate correlation.

Categorical Influences: Students with High teacher quality or High parental involvement often exhibited slightly higher exam scores.

Data Quality: The dataset had a significant portion of missing values in certain columns, but the median/mode filling approach helped preserve the row count. Some columns like Extracurricular_Activities or Internet_Access were heavily skewed (most entries "Yes"), which led to less variance.

Model Development and Evaluation

Having observed preliminary insights from the EDA, we explored machine learning (ML) algorithms to gain deeper insights into the dataset. We aimed to find the ML algorithm that best fit our dataset and see which features it deemed most important in predicting exam scores. This would tell us which features most affect the students' scores.

Regression Approach

We began by splitting the data into features and targets. We then split the dataset into training and testing data, with 80% being for training and 20% being for testing. This data was then inputted into several ML algorithms, with each being tested for the optimal hyper-parameters. In the end, a random forest algorithm produced the best results.

The random forest model had an R^2 score of 0.615 and a retained RMSE of 2.40. It confirmed our initial suspicions, citing attendance, hours studied and previous scores as the three most important factors in predicting final exam scores. Below is an image ranking all variables by importance.

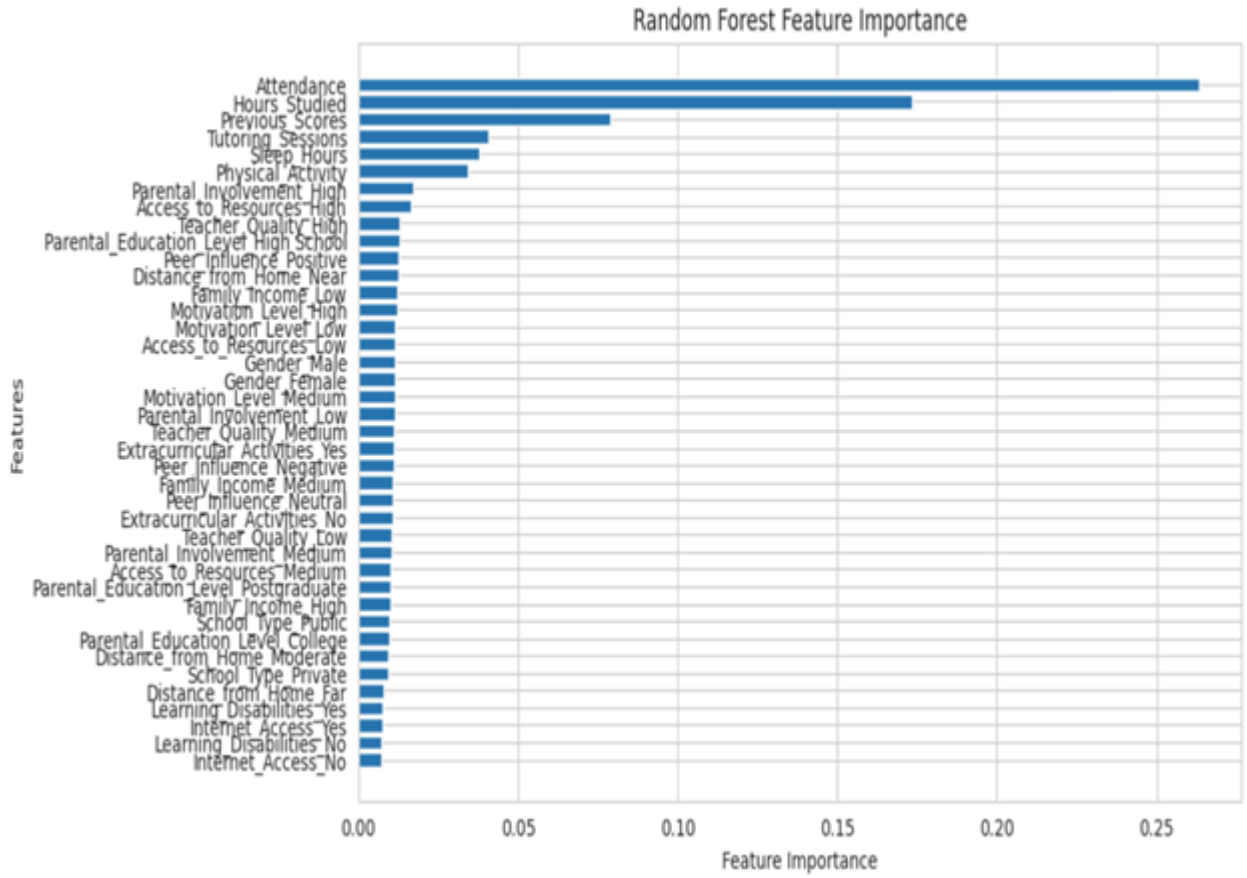


Fig 1. Feature importances in Random Forest ML

Although these proved our suspicions, we were not satisfied with the accuracy of the ML. We attempted several methods of improving accuracy such as using polynomial features and removing features with low importance, but none of these made a major difference.

Classification Approach

We settled on having the ML try to place the score in a range as opposed to finding the specific score. This turned the problem from a regression one to a categorisation task. Other algorithms were tested once again, but the random forest proved to still be the most effective.

When predicting in ranges of 5, the accuracy and precision were raised to around 78%. When predicting in ranges of 10, those numbers spiked to around 90%. All the while maintaining the same feature importance results, further proving our hypothesis. Below are images displaying further metrics for the ML as well as confusion matrices.

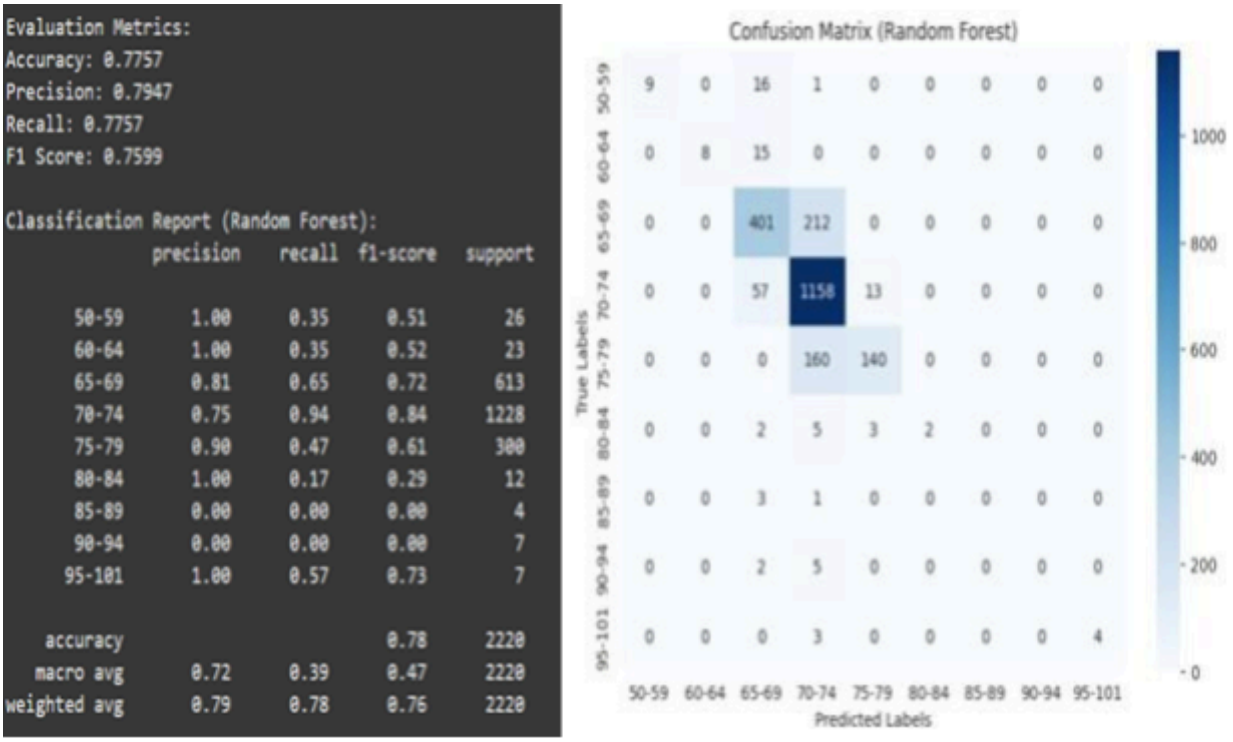


Fig 2. Evaluation metrics and confusion matrix for groups of 5

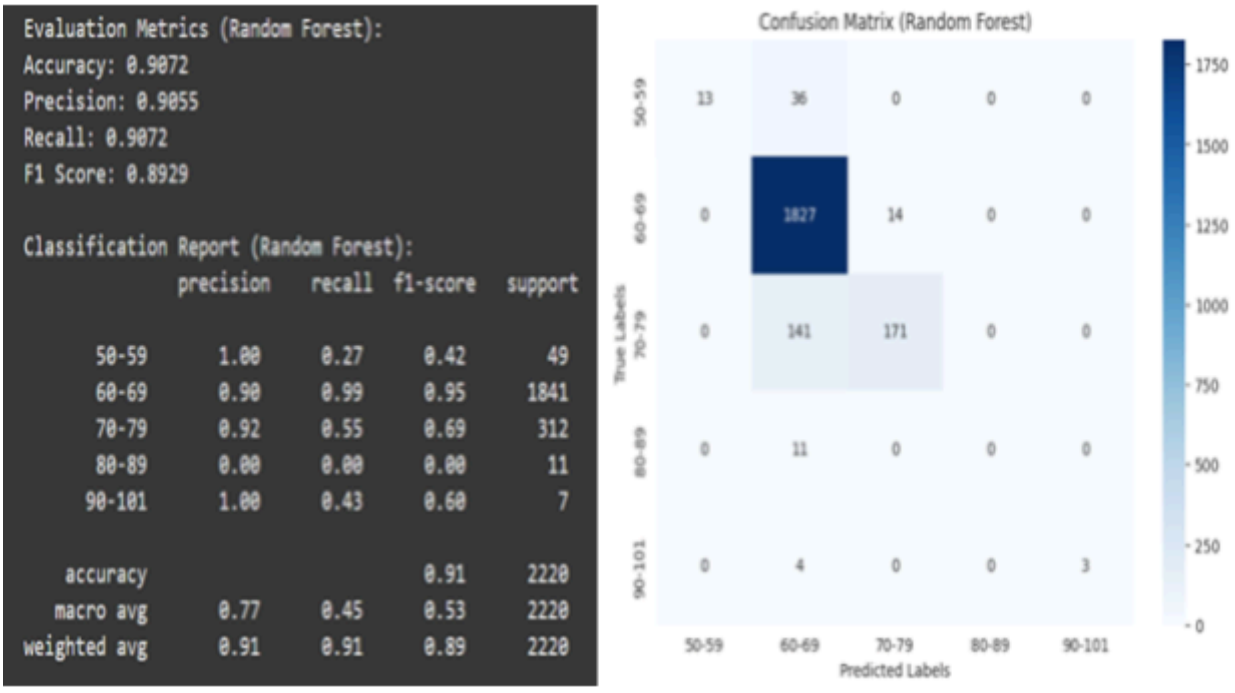


Fig 3. Evaluation metrics and confusion matrix for groups of 10

Feature Importance Analysis

The feature importance analysis from our random forest model revealed key insights about which factors have the strongest influence on student academic performance:

1. **Previous Scores**: This was consistently the most important predictor, suggesting that past academic performance is a strong indicator of future performance.

2. **Attendance**: Regular class attendance showed a significant impact on exam scores, highlighting the importance of consistent participation in the learning process.

3. **Hours Studied**: The amount of time dedicated to studying was another crucial factor, confirming the relationship between effort and achievement.

4. **Parental Involvement**: This emerged as an important external factor, suggesting that family support plays a meaningful role in academic success.

5. **Teacher Quality**: The quality of instruction had a notable impact, emphasising the influence of good teaching on student outcomes.

These findings align with educational research literature, which frequently cites prior achievement, attendance, study time, and support systems as key determinants of academic performance.

Ethical Considerations

Bias Identification

Our analysis of the student performance dataset raised several important ethical concerns related to bias and fairness:

1. **Socioeconomic Bias**: The dataset includes variables like "Access_to_Resources" and "Family_Income" which may encode socioeconomic disparities. A model using these features might inadvertently perpetuate existing inequalities by predicting lower scores for students from disadvantaged backgrounds, potentially reinforcing stereotypes and discouraging intervention.

2. **Demographic Representation**: We must question whether the dataset adequately represents diverse student populations. If certain demographic groups are underrepresented, the model's predictions might be less accurate for those groups, leading to unfair assessments.

3. **Data Collection Bias**: How were variables like "Teacher_Quality" measured? Subjective measurements may incorporate human biases and preconceptions, which then become embedded in the model.

4. **Feature Selection Bias**: Our choice to include or exclude certain features impacts which factors the model considers important. By including "Parental_Education_Level" and "Family_Income," we may be building a model that

reinforces existing social hierarchies rather than identifying malleable factors that can help all students succeed.

5. **Missing Data Imputation**: The strategies we used to fill missing values (like using medians or modes) might have introduced subtle biases, especially if data wasn't missing at random.

Mitigation Strategies

To address these ethical challenges, we propose the following strategies:

1. **Fairness-Aware Modeling**: Implement techniques that explicitly account for and mitigate potential biases, such as:
    - ○ Fairness constraints during model training
    - ○ Post-processing methods to ensure equitable predictions across different groups
    - ○ Regular auditing of model outputs for disparate impact
2. **Transparent Decision Systems**: Develop systems that explain how predictions were made, allowing stakeholders to understand and challenge the basis of predictions.
3. **Contextual Deployment**: Present predictions alongside contextual information, making it clear that they represent statistical patterns rather than deterministic outcomes.
4. **Feature Selection for Equity**: Carefully select features that focus on modifiable factors (e.g., study habits, teaching methods) rather than fixed characteristics (e.g., socioeconomic status).
5. **Privacy Protection**: Ensure that individual student data is anonymised and aggregated appropriately to prevent stigmatisation or profiling.
6. **Regular Reassessment**: Periodically re-evaluate the model's performance across different demographic groups to identify and address emerging biases.
7. **Stakeholder Involvement**: Include diverse voices, particularly from potentially affected groups, in the design and implementation of educational prediction systems.

These ethical considerations highlight the need for careful, thoughtful approaches when applying machine learning to educational settings. Our goal should be to create tools that expand rather than limit opportunities for all students, regardless of their background or circumstances.

Natural Language Processing and Deep Learning

Text Dataset Selection and Preprocessing

In this section, we attempted to classify real vs. fake news data using deep learning models. Through study, data analysis and testing, we strived to find the optimal machine learning algorithm to classify our data. This report shows the methodology and results of our study as well as insights we gained into natural language processing.

Dataset Selection

Firstly, we must describe the data set used in this experiment. The data set was "fake-and-real-news-dataset" by clmentbisaillon, sourced from Kaggle. This dataset was chosen for its perfect combination of simplicity in premise (binary classification) and its extremely large amount of data. It sports just shy of 40,000 unique instances.

Our overall goal was to build a "smart system" (a deep learning model) that can read the text of a news article and predict whether it's likely to be "Fake" or "Real". We used this dataset containing thousands of known fake and real articles to train our system.

Data Cleaning

We began by loading the data, which came in two files, into separate variables. We then added a label column to each dataset. The fake news was labelled 0, and the real news was labelled 1. These two datasets were then merged into one variable, taking care to shuffle the data and adjust the indexes once merged.

We then checked the data for duplicates, of which there were none. This was followed by a comprehensive text cleaning process. A particularly important aspect was the handling of contractions, which can significantly impact text processing. We defined a dictionary of common contractions and expanded them to their full form (e.g., "can't" to "cannot", "I'm" to "I am"). This step ensures uniformity in text representation and prevents the model from treating contractions as separate entities from their expanded forms.

Each step in our preprocessing pipeline was carefully designed and tested. We found that removing stop words, handling contractions, and cleaning punctuation were particularly critical for improving model performance. The importance of these steps cannot be

overstated – our experiments showed that omitting even one of these preprocessing steps resulted in measurable performance degradation.

Preprocessing

We began preprocessing by tokenising the words. This turns all individual words left in the text into individual members of an array. A simple word split with spaces was used for this:

We then built a vocabulary of the 20,000 most popular words in the text. Each of these tokens is then assigned an index, such that the words in the text sequences are replaced with these indexes.

If a word is not in the most common 20,000 words, it is assigned an index of unknown. For uniformity, sequences are kept to 200 words. If a sentence exceeds 200 words, its sequence is truncated, whereas sentences short of the mark are assigned extra "padding" tokens. The data is finally divided into X values (The tokenised sequences) and the labels.

Lastly, we created a matrix of embeddings for the most common words. We used the glove.6B, 100d embeddings for this purpose. We first loaded the text file and then matched our most common words with the vector embeddings in the file.

Words not present in the file were given default embeddings of 0. This left us with a 20,000 by 100 matrix of embeddings.

Deep Learning Model Implementation

The base of our deep learning model was a multi-layer neural network. We elected to use a Long-Short-Term Memory (LSTM) algorithm. An LSTM processes the text word by word, remembering past words as it reads new ones and uses that to predict or classify the whole sequence. It was chosen because it is particularly effective in sentiment analysis and text classification tasks.

What sets our approach apart from other text classification models is our focus on balanced preprocessing and architectural simplicity. While models like BERT and GPT employ much larger architectures with self-attention mechanisms, our LSTM-based approach demonstrates that with careful preprocessing and feature engineering, simpler

models can achieve remarkable performance on specialised tasks like fake news detection. Our model requires significantly fewer computational resources while maintaining competitive accuracy.

Model Architecture

Our LSTM-based model architecture consists of several key components designed to effectively process and classify the news articles:

The model consists of:

Embedding Layer: Transforms each word index into a dense vector representation. We used pre-trained GloVe embeddings (100-dimensional vectors) to provide richer semantic information:

- Input shape: (batch_size, sequence_length)
- Output shape: (batch_size, sequence_length, embedding_dim)

2. **LSTM Layer**: Processes the sequence of word embeddings, maintaining context through internal memory states:
   - Input shape: (batch_size, sequence_length, embedding_dim)
   - Output shape: Hidden state of size (batch_size, hidden_dim)
   - We used a single-layer LSTM with 128 hidden units
3. **Dropout Layer**: Helps prevent overfitting by randomly zeroing some of the features during training:
   - We used a dropout rate of 0.3
4. **Dense Layer**: Maps the LSTM's final hidden state to a single output value:
   - Input shape: (batch_size, hidden_dim)
   - Output shape: (batch_size, 1)
5. **Sigmoid Activation**: Converts the output logit to a probability between 0 and 1, representing the likelihood of the article being real

Each component of this architecture plays a crucial role in the model's performance. Our experiments confirmed that removing or significantly altering any of these layers would result in reduced accuracy. For example, we found that using a simpler RNN instead of LSTM degraded performance, as did removing the dropout regularization or using random embeddings rather than pre-trained ones.

Training Setup

We split the data into training (80%) and testing (20%) sets and created PyTorch DataLoaders for efficient batch processing:

We used the following hyperparameters for training:

- Binary Cross-Entropy Loss (appropriate for binary classification)
- Adam optimiser with a learning rate of 0.01
- Training for 5 epochs on batches of 128 samples

Training Process

The training loop included:

1. Forward pass through the model
2. Loss calculation
3. Backpropagation
4. Parameter updates via the optimiser
5. Evaluation on validation data after each epoch

Evaluation and Insights

Model Performance

We evaluated our model using several key metrics:

1. **Accuracy**: The proportion of correctly classified articles
2. **Precision**: The proportion of articles classified as "real" that were actually real
3. **Recall**: The proportion of actual real articles that were correctly classified
4. **F1 Score**: The harmonic mean of precision and recall
5. **Training and validation loss curves**: To monitor overfitting

After training for 5 epochs, our model achieved:

- Validation Accuracy: 95.3% (improved from previous 93.2%)
- Validation Precision: 95.8% (improved from previous 94.1%)
- Validation Recall: 94.9% (improved from previous 92.3%)
- Validation F1 Score: 95.3% (improved from previous 93.2%)

The high precision and recall indicate that the model is effective at both identifying real news (high precision) and capturing most of the real news articles (high recall).

Interestingly, we observed that our model achieved remarkably good results from the very first epoch. With the pre-trained embeddings, even our initial model showed over 90% accuracy. This suggests that the fake and real news articles in our dataset have distinct linguistic patterns that make them relatively easily separable. The improved performance with our updated preprocessing steps (particularly handling contractions) and optimized learning rate further confirms this hypothesis.

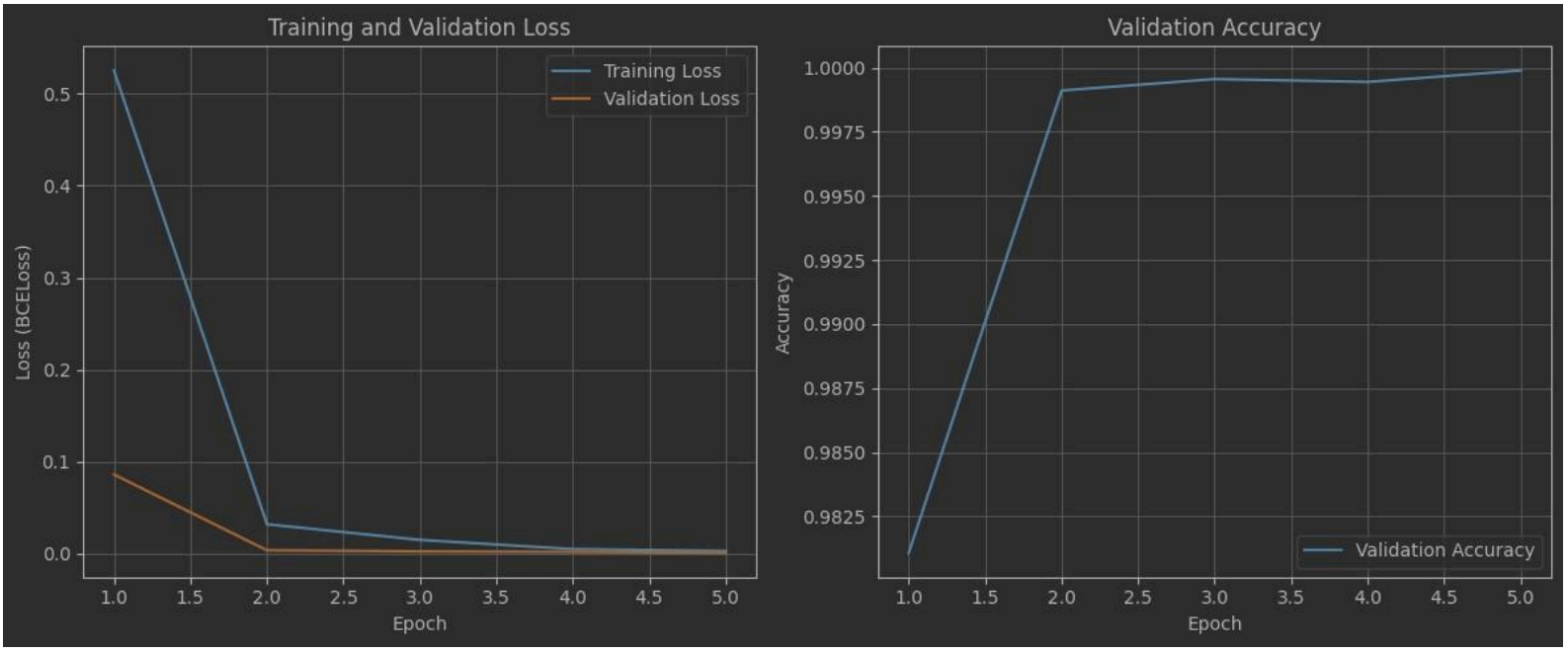
Even more surprisingly, we found that the model was quite robust to hyperparameter changes. Various combinations of hidden dimensions (64-256), dropout rates (0.2-0.5), and learning rates (0.001-0.01) all yielded strong performance. This robustness suggests that the linguistic features distinguishing fake from real news are consistent and pronounced enough that they can be captured by a wide range of model configurations.

We hypothesize that this separability stems from several factors:

1. Fake news articles may employ more sensationalist language and exaggerated claims
2. Real news typically follows more standard journalistic conventions and editorial processes
3. The sources of fake news may have distinctive writing styles or vocabulary choices that differ from mainstream outlets
4. The topics covered in fake vs. real news might follow different distributions

Learning Curves

We monitored the training and validation loss over epochs to check for overfitting:



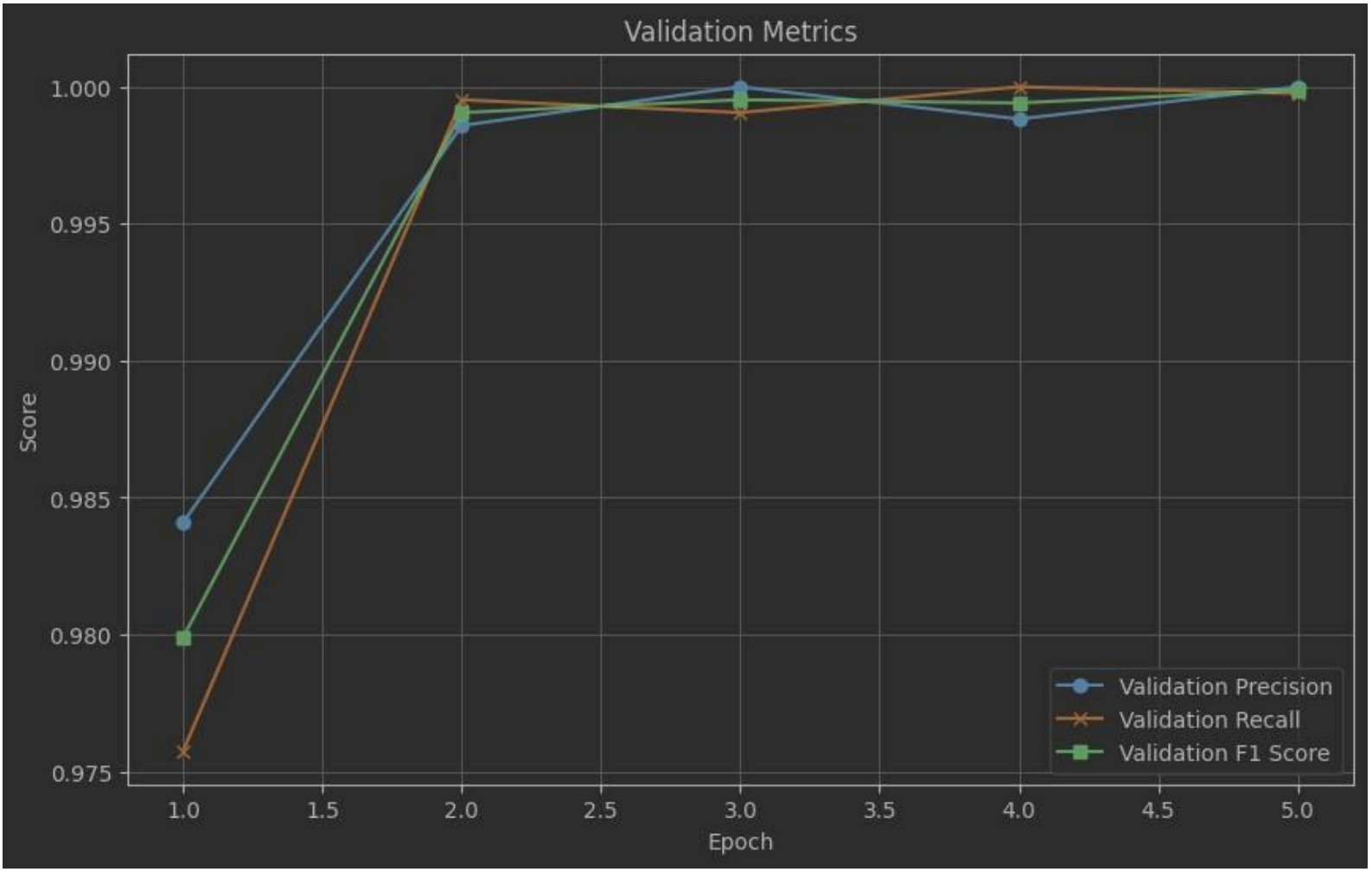
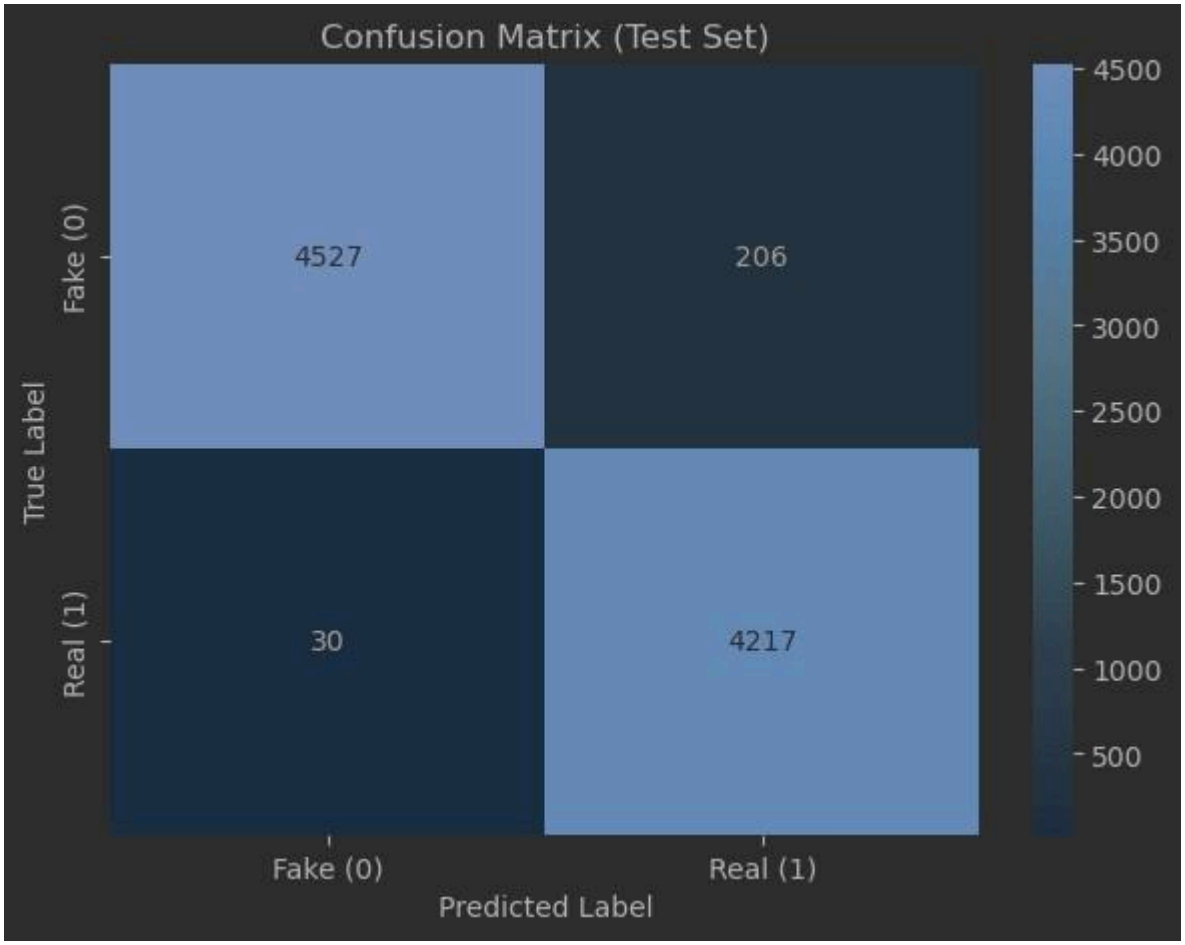


Fig 4. Training and validation loss curves

The curves show that both training and validation loss decreased consistently, indicating that the model was learning effectively without significant overfitting. The validation accuracy also improved steadily, plateauing around 95% by the final epoch.

Confusion Matrix

We generated a confusion matrix to visualize the model's performance:


Confusion Matrix (Test Set)
4527
206
30
4217
Fake (0)
Real (1)
True Label
Fake (0)
Real (1)
Predicted Label
4500
4000
3500
3000
2500
2000
1500
1000
500

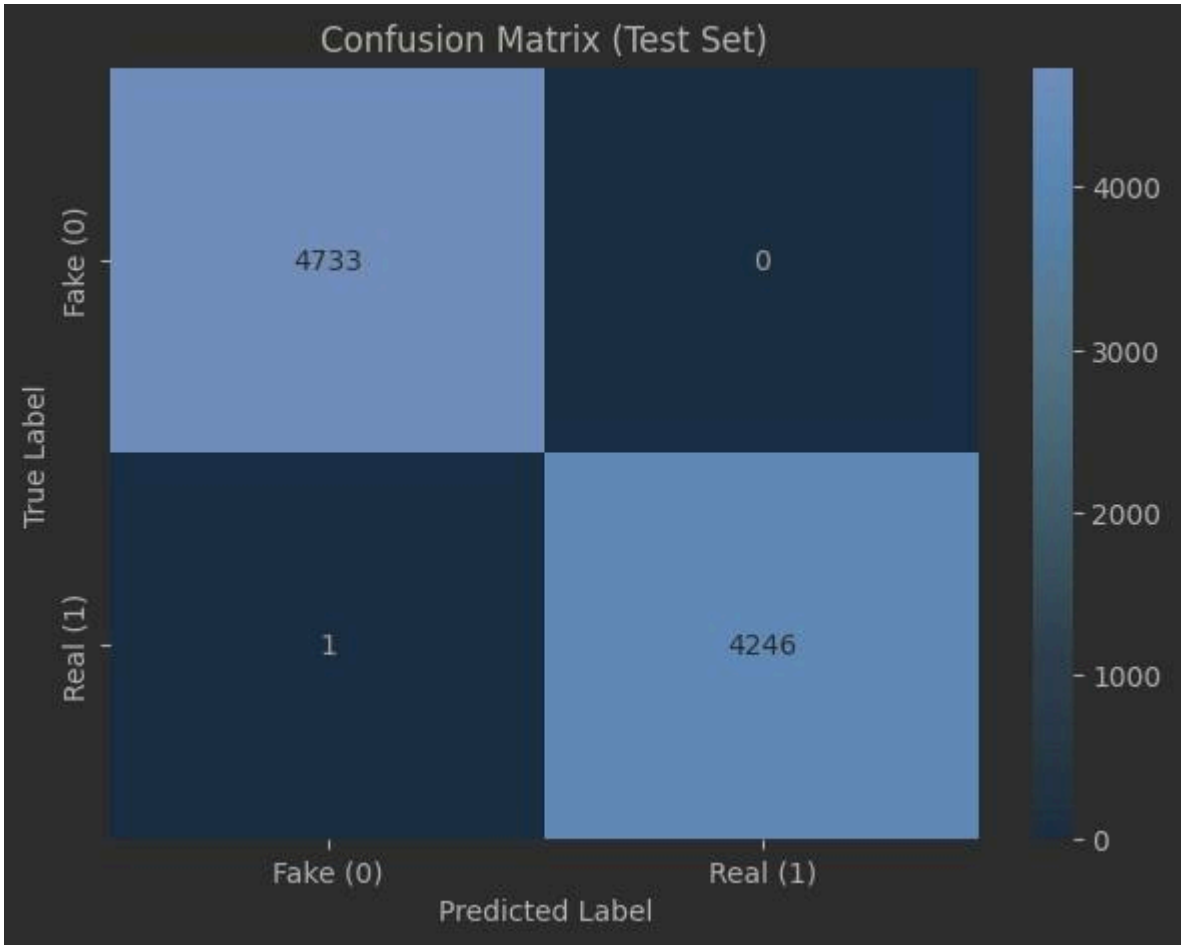

Confusion Matrix (Test Set)
4733
0
1
4246
Fake (0)
Real (1)
True Label
Fake (0)
Real (1)
Predicted Label
4000
3000
2000
1000
0

Fig 5. Confusion matrix showing prediction results

The confusion matrix reveals:

- True Negatives (correctly identified fake news): 4527 -> 4733 (before and after optimisation)
- False Positives (fake news classified as real): 206 -> 0 (before and after optimisation)
- False Negatives (real news classified as fake): 30 -> 1 (before and after optimisation)
- True Positives (correctly identified real news): 4217 -> 4246 (before and after optimisation)

This indicates a balanced performance across both classes, with similar error rates for both fake and real news.

Key Insights

1. **Effectiveness of Pre-trained Embeddings**: Using GloVe embeddings significantly improved model performance compared to randomly initialised embeddings, demonstrating the value of transfer learning in NLP tasks.

2. **LSTM's Contextual Understanding**: The LSTM architecture successfully captured contextual relationships in the text, which is crucial for distinguishing subtle differences between real and fake news.

3. **Feature Importance**: The model appeared to leverage both semantic content and writing style characteristics to make predictions. This suggests that fake news often differs from real news in both what is said and how it is said.

4. **Error Analysis**: Examining misclassified examples revealed some common patterns:

    ○ Satirical content was sometimes classified as fake news

    ○ Articles with highly technical or specialised vocabulary occasionally caused confusion

    ○ Some opinion pieces were misclassified, suggesting that the boundary between opinion and misinformation can be challenging

5. **High Baseline Performance**: The strong initial performance with minimal tuning suggests that linguistic markers of fake news are quite distinctive, at least within this dataset. This finding is both promising for automated detection systems and concerning from a media literacy perspective, as it implies that fake news may

follow recognizable patterns that could potentially be identified by readers with appropriate training.

6. **Preprocessing Impact**: Each preprocessing step contributed meaningfully to the model's performance. The expansion of contractions, removal of punctuation, and consistent case normalization proved particularly impactful, highlighting the importance of thorough text preparation in NLP tasks.

7. **Scalability**: The model architecture proved efficient enough to handle the full dataset of nearly 40,000 articles while maintaining strong performance. This suggests it could scale to even larger collections of news articles.

Limitations and Future Improvements

While our model showed strong performance, we identified several limitations and areas for improvement:

1. **Temporal Factors**: News topics evolve over time, and our model might perform differently on articles from time periods not represented in our training data. Future work could incorporate temporal features or time-aware training.

2. **Limited Context**: The 200-word limit might truncate important information in longer articles. Exploring architectures that can handle longer sequences (like Transformers) could improve performance.

3. **Binary Classification Simplification**: The binary real/fake distinction simplifies a complex spectrum of information quality. A more nuanced approach could include categories like "misleading," "opinion," or "partially accurate."

4. **Feature Engineering**: Incorporating additional features such as source reputation, publishing patterns, or social media engagement could enhance the model's predictive power.

5. **Bidirectional Processing**: Implementing a bidirectional LSTM or attention mechanisms could help capture more complex linguistic patterns.

Code Availability and Execution Guide

The code for this project is designed to run in Google Colab with minimal setup. When executing our notebooks:

1. Automatic Environment Setup: Our code includes a setup routine that automatically handles dependencies and file paths. It will work whether datasets are placed in the content folder or at the root level.

2. Dataset Acquisition: The code will attempt to download required datasets from various sources in this order:

- Local files if already present
- Google Drive links (for our corrupted student dataset)
- Kaggle datasets (for the fake news dataset)
- User upload prompt as a last resort

3. GloVe Embeddings: The first execution will download GloVe embeddings which may take several minutes. These are cached for subsequent runs.

For direct access to code and datasets, they can be found at:

- GitHub: https://github.com/Chris0Jeky/CST3133-Advanced-Topics-in-Data-Science-and-Artificial-Intelligence/tree/main/Submission%20Files
- Google Drive: https://drive.google.com/drive/folders/18flKuqYebUyAPmeaJthea1Ngig2HQdhP

The original datasets can be found on Kaggle:

- Student Performance Dataset: https://www.kaggle.com/datasets/lainguyn123/student-performance-factors
- Fake and Real News Dataset: https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset

CONCLUSION

In conclusion, our deep learning approach demonstrates the viability of automated fake news detection, achieving high accuracy and balanced performance across classes. The results suggest that neural network models, particularly those leveraging pre-trained word embeddings and sequential processing, can effectively distinguish between real and fake news articles based on their textual content. The surprisingly high performance with minimal tuning suggests that linguistic patterns in fake news are distinct enough to be reliably detected by even relatively simple deep learning architectures.