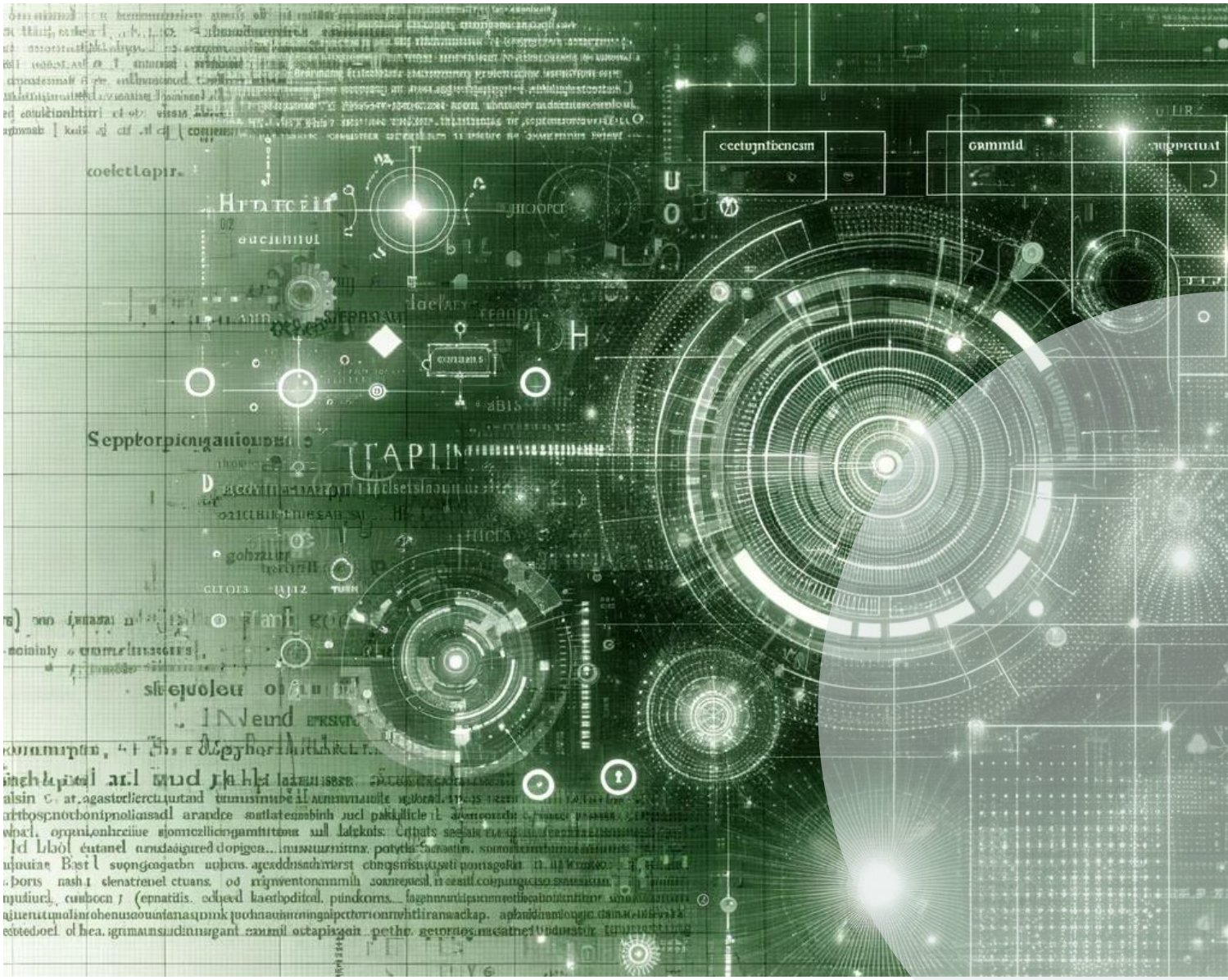
CAPSTONE PROJECT



Developing a Machine Learning Based Classifier for Identifying Exceptional Facts in Text

ALISHBAH FAHAD

1001924185

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to IDIR LAB for their invaluable support and resources throughout the project. I would also like to extend my special thanks to Dr. Li for providing expert guidance and continuous mentorship. In addition, I appreciate Nasim’s consistent help and contributions, which played a significant role in the success of the project.

ABSTRACT

The field of data science requires the ability to sort through large amounts of text and identify important information, known as "exceptional facts". Our report outlines the development of a machine learning-based classifier that can identify these facts within text. We used the advanced natural language processing capabilities of the DistilBERT model to build a classifier that detects exceptional facts based on specific criteria. We carefully acquired, pre-processed, and annotated the data, and then trained and evaluated our model using stratified K-Fold cross-validation. The classifier achieved promising results, with an average accuracy of 77.27% across the test folds. Our report not only details the technical processes involved, but also explores the broader implications of this work in academic research, journalism, and decision-making, highlighting the importance of identifying exceptional facts in the digital age.

INTRODUCTION

Background and Significance

In the digital age, the sheer volume of textual data available presents both an opportunity and a challenge. One of the key challenges lies in identifying pieces of information known as "exceptional facts." An exceptional fact is a statement that highlights rare characteristics of a particular entity within a larger group, consisting of three main components: the entity of interest, the context, and the qualifying attributes. For example, consider the statement "Denzel Washington won the Best Actor award." Here, Denzel Washington is the entity of interest; the context is all Best Actor award winners, and the qualifying attributes are based on ethnicity. The exceptional fact in this instance is that among all the award winners, Denzel Washington stands out as one of the few African Americans to win.

The identification of such facts has traditionally required expert human analysis, a method that is not scalable to the vast amounts of data generated every day. This project is significant because it seeks to automate the extraction of exceptional facts from large datasets. These facts are important as they provide insights that can influence academic research, inform journalistic reporting, and guide strategic business decisions.

Problem Statement

Although exceptional facts are essential in information analysis, their identification process remains labor-intensive, subjective, and prone to errors. With the increasing amount of information processed every day, there is a clear need for an automated solution that can accurately and efficiently perform this task. However, current automated text analysis tools mainly focus on general patterns and sentiment analysis, and they give less importance to identifying the rare, nuanced pieces of information that constitute exceptional facts.

Objectives

The main goal of this project is to create a classifier using machine learning that can identify exceptional facts within large textual datasets. By utilizing a more concise version of the BERT model known as DistilBERT, this project aims to:

- Automate the identification of exceptional facts with high accuracy and completeness.
- Enable the processing of vast amounts of textual data, reducing the required time and resources considerably.
- Offer a tool that can help researchers, journalists, and decision-makers to recognize important pieces of information that might otherwise be overlooked.

Scope and Limitations

The purpose of this project is to develop and evaluate a DistilBERT-based classifier that can identify exceptional facts within textual data. The project covers the conceptualization, design, development, and preliminary testing of the classifier on a dataset comprising sentences from Wikipedia, each annotated for the presence or absence of exceptional facts.

However, it is essential to note that this project has some limitations. Firstly, the dataset used for training and testing is relatively small, which may limit the classifier's ability to learn from a more extensive and diverse set of examples. This limitation could potentially impact the model's generalizability to other applications. Secondly, due to constraints in computational resources, the project could not undertake extensive hyperparameter tuning, which is often critical for optimizing machine learning model performance. Lastly, the project's timeline was limited, which restricted the iterative model improvement that could be performed. While these limitations constrain the current phase of the project, they also highlight areas for future work and improvement.

LITERATURE REVIEW

The automation of textual analysis through machine learning has been a subject of increasing interest in various fields, including data science, computational linguistics, and artificial intelligence. The literature reveals a strong trend towards developing algorithms that can efficiently process and interpret large amounts of textual data with minimal human intervention.

Historical Context and Evolution

The search for automating the identification of important information in text goes back to the early days of information retrieval systems. As early as the 1950s, Luhn's innovative

work on keyword-based methods laid the foundation for later developments in the field. With the rise of the internet and digital storage, text classification became a priority to effectively organize and categorize vast digital text repositories.

Machine Learning and NLP in Text Classification

The field of text classification underwent a significant transformation with the advent of machine learning. In the late 20th century, there was a noticeable shift towards machine learning-based models from the previous rule-based approach. Early algorithms applied to text classification tasks included Support Vector Machines (SVMs), Decision Trees, and Naive Bayes classifiers. However, the introduction of deep learning brought about a significant improvement in the ability to capture complex patterns in data. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks, have become widely adopted for their ability to handle sequential data and capture contextual information.

BERT and its Derivatives

The BERT (Bidirectional Encoder Representations from Transformers) model was a significant development in natural language processing (NLP). It introduced a new method of pre-training language representations that could be fine-tuned for a variety of tasks. BERT's bidirectional training and transformer architecture allowed for a deeper understanding of context, resulting in new NLP benchmarks.

DistilBERT, a streamlined version of BERT, retains most of the original model's effectiveness but with fewer parameters, making it more efficient and faster. This makes DistilBERT particularly suitable for applications where computational resources are a concern. The literature suggests that DistilBERT achieves remarkable results in text classification tasks, making it an ideal choice for our project's objectives.

METHODOLOGY

This section describes the methodology used for developing a DistilBERT-based classifier to identify exceptional facts in text. The process involves multiple stages, from data preparation to model training and evaluation.

Data Preparation

The initial phase of the project involved collecting and preparing the dataset, a necessary step for any machine learning project.

1. **Sentence Collection:**
    - The sentences were exclusively collected from Wikipedia, providing a rich and diverse source of information.
    - A total of 55 keywords, identified through the analysis of existing exceptional facts, were used to guide the search and extraction process. This targeted approach ensured the relevance and quality of the collected sentences.

2. **Data Annotation:**
    - The annotation process was undertaken solely, bringing in a consistent standard of evaluation across all data.
    - Each sentence was carefully labeled as 'yes' (containing an exceptional fact) or 'no' (not containing an exceptional fact) based on predefined criteria.

3. **Dataset Characteristics:**
    - The final dataset comprised 288 sentences, balanced between the 'yes' and 'no' classes.
    - This relatively small dataset size is one of the project's limitations, as it may impact the classifier's learning capacity and generalizability.

Model Development

The core of the project was the development of the classifier using the DistilBERT model, a process that involved several steps:

1. **Tokenizer Initialization:**
    - The *DistilBertTokenizerFast* was employed for tokenizing the sentences, converting them into a format suitable for the DistilBERT model.

2. **Dataset Splitting:**
    - The dataset was split into training and testing sets, with 85% of the data used for training and the remaining 15% reserved for testing.
3. **Model Initialization:**
    - *DistilBertForSequenceClassification,* a pre-trained DistilBERT model, was initialized for the task of sequence classification.
4. **Training Process:**
    - A Stratified 5-Fold Cross-Validation method was employed to ensure a robust evaluation.
    - The model was trained for 3 epochs, balancing the need for adequate learning without overfitting.
    - A learning rate of 5e-5 was set for the training process.
    - The batch size was fixed at 16, optimizing the computational resources available.
    - The AdamW optimizer was used, a variant of the Adam optimizer known for its effectiveness in NLP tasks.

Model Evaluation

The evaluation of the model focused on its ability to accurately classify sentences as containing or not containing exceptional facts.

1. **Evaluation Metrics:**
    - The model's performance was assessed using accuracy, precision, recall, and F1 score.
    - Cross-validation results provided insights into the model's consistency and reliability.
2. **Testing and Validation:**
    - Following training, the model was evaluated on unseen data from the testing set.
    - The evaluation results, including a confusion matrix, were used to gauge the model's effectiveness in real-world scenarios.

RESULTS

The results section presents the outcomes of the classifier's training and testing, providing insights into its performance in identifying exceptional facts in text.

Training Results

1. **Cross-Validation Performance:**

- The model demonstrated varying levels of accuracy across different folds, reflecting its responsiveness to the nuances in the dataset.
- Notable metrics from the training phase (averaged across folds) included:
    - Accuracy: Ranging from 71.43% to 100%, indicating the classifier's varying effectiveness in different subsets of the dataset.
    - Precision and Recall: These metrics varied across folds, reflecting the model's ability to correctly identify positive and negative classes.

2. **Insights from Training:**

    - The balanced nature of the dataset in terms of class distribution ('yes' and 'no' classes) contributed to a more unbiased learning process.
    - The limitations in dataset size and the absence of extensive hyperparameter tuning were observed to potentially affect the model's performance.

Testing Results

The testing phase provided an evaluation of the classifier's performance on an unseen dataset, offering a realistic assessment of its effectiveness.

1. **Overall Testing Performance:**

    - The classifier achieved an average test accuracy of 77.27%.
    - Precision, recall, and F1 score metrics also indicated a solid performance, with average values of 76.47%, 76.32%, and 76.10% respectively.

2. **Confusion Matrix Analysis:**
    - The model correctly identified the absence of exceptional facts 72 times (true negatives).
    - It accurately detected the presence of exceptional facts 64 times (true positives).
    - The model missed 20 exceptional facts, marking them incorrectly as non-exceptional (false negatives).
    - It incorrectly identified 20 non-exceptional facts as exceptional (false positives).
    - This data from the confusion matrix shows that the classifier maintained a balance in identifying both classes, with an equal number of false positives and negatives.

DISCUSSION

The Discussion section interprets the results of the classifier and explores their implications in the broader context of machine learning and text analysis. It also addresses the limitations of the study and suggests avenues for future research.

Interpretation of Results

1. **Performance of the Classifier:**
    - The classifier demonstrated a promising performance with an average accuracy of 77.27% in testing. This indicates a strong potential for using machine learning to automate the identification of exceptional facts.
    - The balance of precision, recall, and F1 scores suggests that the classifier was able to effectively balance false positives and false negatives, a critical aspect in the context of exceptional fact identification.

2. **Implications for Text Analysis:**
    - The project showcases the capability of machine learning models, particularly DistilBERT, in extracting nuanced information from text. This opens up possibilities for automating complex analytical tasks in various domains, such as journalism, research, and business intelligence.

Limitations and Challenges

1. **Dataset Constraints:**
   - The primary limitation was the dataset size. A larger and more diverse dataset could potentially enhance the classifier's performance and generalizability.
   - The sole annotation, while ensuring consistency, might also introduce a degree of subjective bias in data labeling.

2. **Computational Limitations:**
   - The inability to conduct extensive hyperparameter tuning due to computational and time constraints limited the optimization of the classifier. Future iterations of the project with more computational resources could explore this further.

Future Research Directions

1. **Dataset Expansion:**
   - Expanding the dataset, both in size and diversity, would be a crucial step in improving the model. A larger dataset might also include more complex and varied examples of exceptional facts.

2. **Algorithmic Improvements:**
   - Future work could include extensive hyperparameter tuning and exploring other NLP models or ensemble methods to enhance the classifier's performance.

3. **Application in Different Contexts:**
   - Applying the classifier to different domains and in other languages could be explored to assess its adaptability and utility in various settings.

4. **Addressing Subjectivity in Annotation:**
   - Employing multiple annotators and establishing more rigorous annotation guidelines could help mitigate the subjectivity in data labeling.

CONCLUSION

The objective of this project was to develop a machine learning classifier capable of identifying exceptional facts in text, a task with significant implications in fields like journalism, academic research, and data analysis. This report has documented the process of developing such a classifier using the DistilBERT model, from the initial data preparation phase through to the model's training, evaluation, and testing.

The classifier achieved an average test accuracy of 77.27%, a promising result given the constraints under which it was developed. This demonstrates the feasibility and potential effectiveness of using machine learning models for the nuanced task of identifying exceptional facts in text.

By automating the process of identifying exceptional facts, this project contributes to the broader goal of enhancing the efficiency and accuracy of information extraction from large textual datasets. It showcases how advanced NLP techniques can be applied to specific, high-value tasks in information processing.

The project faced limitations in terms of the size of the dataset and the computational resources available. These limitations highlight the importance of resource allocation in machine learning projects and point to the need for further research and development.

Expanding the dataset, conducting extensive hyperparameter tuning, and exploring the model's application in different contexts are identified as key areas for future work. Such efforts could significantly improve the model's performance and its applicability to a wider range of text analysis tasks.

In conclusion, this project represents a step forward in the field of automated text analysis. It has successfully demonstrated the application of a DistilBERT-based machine learning model to identify exceptional facts in text, laying the groundwork for future research and development in this area.

REFERENCES

1. Ambalavanan, A. K., & Devarakonda, M. V. (2020). Using the contextual language model BERT for multi-criteria classification of scientific articles. Journal of Biomedical Informatics, 112.
2. Analysis of BERT and DistilBERT for text classification in English and Brazilian Portuguese. (2022).
3. Muffo, M., & Bertino, E. (2022). A Fine-Tuned BERT-Based Transfer Learning Approach for Text Classification. PMC Journal List, Journal of Healthcare Engineering.
4. Zhang, G., & Li, C. (2018). Maverick: A System for Discovering Exceptional Facts from Knowledge Graphs. Proceedings of the VLDB Endowment (PVLDB), 11(12), 1934-1937.

APPENDICES

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.tokenize import word_tokenize
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
from torch.utils.data import Dataset, DataLoader, SubsetRandomSampler
import torch
from transformers import DistilBertForSequenceClassification
from torch.optim import AdamW
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import numpy as np
import random
```

Figure 1 Importing Necessary Libraries

```
# Load the dataset
file_path = r"C:\Users\alish\OneDrive\Documents\Alishbah\DASC5309_DATA SCIENCE CAPSTONE PROJECT\dataset\classification.xlsx"
data = pd.read_excel(file_path)

# Display the first few rows of the dataset
data.head()
```

	Sentence	Class
0	Chen Kaige followed up the Unprecedented succe...	yes
1	Promoted by Wakefield Poole with an advertisin...	yes
2	the Jean Renoir film "La Grande Illusion", an ...	yes
3	Sing Your Song shows not only Harry Belafonte'...	no
4	What makes Jennifer Connelly so Remarkable isn...	no

Figure 2 Loading the Dataset

```python
# Basic information about the dataset
total_entries = data.shape[0]
class_distribution = data['Class'].value_counts()
missing_values = data.isnull().sum().sum()  # Sum of all missing values across all columns

# Formatting the output
output = f"Total Entries: {total_entries}\n"
output += "Class Distribution:\n"
for class_name, count in class_distribution.items():
    output += f"'{class_name}': {count} entries\n"
output += f"Missing Values: {missing_values}"

print(output)
```

```
Total Entries: 288
Class Distribution:
'yes': 150 entries
'no': 138 entries
Missing Values: 0
```

Figure 3 Dataset Overview

```python
# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Initialize tokenizer
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

# Convert class labels to integers
label_to_id = {'yes': 1, 'no': 0}
data['Label'] = data['Class'].map(label_to_id)

# Split the dataset into training and testing sets
train_data, test_data = train_test_split(data, test_size=0.15, random_state=seed)

# Tokenize the training data
train_encodings = tokenizer(list(train_data['Sentence']), truncation=True, padding=True, return_tensors="pt")
train_labels = torch.tensor(train_data['Label'].values)

# Custom dataset class
class TextDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx].clone().detach() for key, val in self.encodings.items()}
        item['labels'] = self.labels[idx]
        return item

    def __len__(self):
        return len(self.labels)

# Convert to torch datasets
train_dataset = TextDataset(train_encodings, train_labels)

# Define KFold Cross-Validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)

# Load the DistilBERT model
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
model.to(device)

# Training parameters
optimizer = AdamW(model.parameters(), lr=5e-5)
num_epochs = 3
```

Figure 4 DistilBERT Model Initialization and Training Setup

```python
# Start stratified cross-validation for training
for fold, (train_ids, val_ids) in enumerate(kfold.split(train_encodings['input_ids'], train_labels)):
    print(f'TRAINING FOLD {fold}')
    print('--------------------------------')

    # Data loaders for training and validation sets
    train_loader = DataLoader(train_dataset, batch_size=16, sampler=SubsetRandomSampler(train_ids))
    val_loader = DataLoader(train_dataset, batch_size=16, sampler=SubsetRandomSampler(val_ids))

    # Training loop for the current fold
    for epoch in range(num_epochs):
        model.train()
        for batch in train_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            loss = outputs.loss
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        print(f"Epoch {epoch+1}/{num_epochs} completed.")

    # Evaluation for the current fold
    model.eval()
    predictions = []
    real_values = []
    with torch.no_grad():
        for batch in val_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            predictions.extend(torch.argmax(logits, dim=1).tolist())
            real_values.extend(batch['labels'].tolist())

    # Calculate metrics for the current fold
    accuracy = accuracy_score(real_values, predictions)
    precision = precision_score(real_values, predictions)
    recall = recall_score(real_values, predictions)
    f1 = f1_score(real_values, predictions)
    print(f"Metrics in fold {fold}: Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")

print("Cross-validation training completed.")
```

Figure 5 Model Training and Evaluation Loop Code

```
TRAINING FOLD 0
---------------------------------
Epoch 1/3 completed.
Epoch 2/3 completed.
Epoch 3/3 completed.
Metrics in fold 0: Accuracy: 0.7142857142857143, Precision: 0.7142857142857143, Recall: 0.7692307692307693, F1 Score: 0.7407407407407408
TRAINING FOLD 1
---------------------------------
Epoch 1/3 completed.
Epoch 2/3 completed.
Epoch 3/3 completed.
Metrics in fold 1: Accuracy: 0.9183673469387755, Precision: 0.8666666666666667, Recall: 1.0, F1 Score: 0.9285714285714286
TRAINING FOLD 2
---------------------------------
Epoch 1/3 completed.
Epoch 2/3 completed.
Epoch 3/3 completed.
Metrics in fold 2: Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0
TRAINING FOLD 3
---------------------------------
Epoch 1/3 completed.
Epoch 2/3 completed.
Epoch 3/3 completed.
Metrics in fold 3: Accuracy: 0.9795918367346939, Precision: 0.9629629629629629, Recall: 1.0, F1 Score: 0.9811320754716981
TRAINING FOLD 4
---------------------------------
Epoch 1/3 completed.
Epoch 2/3 completed.
Epoch 3/3 completed.
Metrics in fold 4: Accuracy: 1.0, Precision: 1.0, Recall: 1.0, F1 Score: 1.0
Cross-validation training completed.
```

Figure 6 Model Training Output Log

```python
# Tokenize the test data
test_encodings = tokenizer(list(test_data['Sentence']), truncation=True, padding=True, return_tensors="pt")
test_labels = torch.tensor(test_data['Label'].values)

# Convert to torch dataset
test_dataset = TextDataset(test_encodings, test_labels)

# Start stratified cross-validation for testing
for fold, (test_ids, _) in enumerate(kfold.split(test_encodings['input_ids'], test_labels)):
    print(f'TESTING FOLD {fold}')
    print('--------------------------------')

    # Data loaders for test set
    test_loader = DataLoader(test_dataset, batch_size=16, sampler=SubsetRandomSampler(test_ids))

    # Evaluation loop for the current fold
    model.eval()
    fold_predictions = []
    fold_real_values = []
    with torch.no_grad():
        for batch in test_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            fold_predictions.extend(torch.argmax(logits, dim=1).tolist())
            fold_real_values.extend(batch['labels'].tolist())

    # Calculate and store the confusion matrix for the current fold
    fold_confusion_matrix = confusion_matrix(fold_real_values, fold_predictions, labels=[0, 1])
    confusion_matrices.append(fold_confusion_matrix)

    # Calculate and store metrics for the current fold
    fold_accuracy = accuracy_score(fold_real_values, fold_predictions)
    fold_precision = precision_score(fold_real_values, fold_predictions)
    fold_recall = recall_score(fold_real_values, fold_predictions)
    fold_f1 = f1_score(fold_real_values, fold_predictions)

    test_accuracies.append(fold_accuracy)
    test_precisions.append(fold_precision)
    test_recalls.append(fold_recall)
    test_f1_scores.append(fold_f1)

    print(f"Metrics in test fold {fold}: Accuracy: {fold_accuracy}, Precision: {fold_precision}, Recall: {fold_recall}, F1 Score: {fold_f1}")
```

Figure 7 Testing and Evaluation Code

```
TESTING FOLD 0
--------------------------------
Metrics in test fold 0: Accuracy: 0.8, Precision: 0.7368421052631579, Recall: 0.875, F1 Score: 0.7999999999999999
TESTING FOLD 1
--------------------------------
Metrics in test fold 1: Accuracy: 0.7428571428571429, Precision: 0.75, Recall: 0.7058823529411765, F1 Score: 0.7272727272727272
TESTING FOLD 2
--------------------------------
Metrics in test fold 2: Accuracy: 0.7714285714285715, Precision: 0.8, Recall: 0.7058823529411765, F1 Score: 0.7500000000000001
TESTING FOLD 3
--------------------------------
Metrics in test fold 3: Accuracy: 0.7714285714285715, Precision: 0.7368421052631579, Recall: 0.8235294117647058, F1 Score: 0.7777777777777778
TESTING FOLD 4
--------------------------------
Metrics in test fold 4: Accuracy: 0.7777777777777778, Precision: 0.8, Recall: 0.7058823529411765, F1 Score: 0.7500000000000001
Average Test Accuracy: 0.7726984126984127, Standard Deviation: 0.0182477560167618
Average Test Precision: 0.7647368421052633, Standard Deviation: 0.029190367629183294
Average Test Recall: 0.7632352941176471, Standard Deviation: 0.07210382748312505
Average Test F1 Score: 0.761010101010101, Standard Deviation: 0.02522705786503174
```

Figure 8 Test Fold Evaluation Results

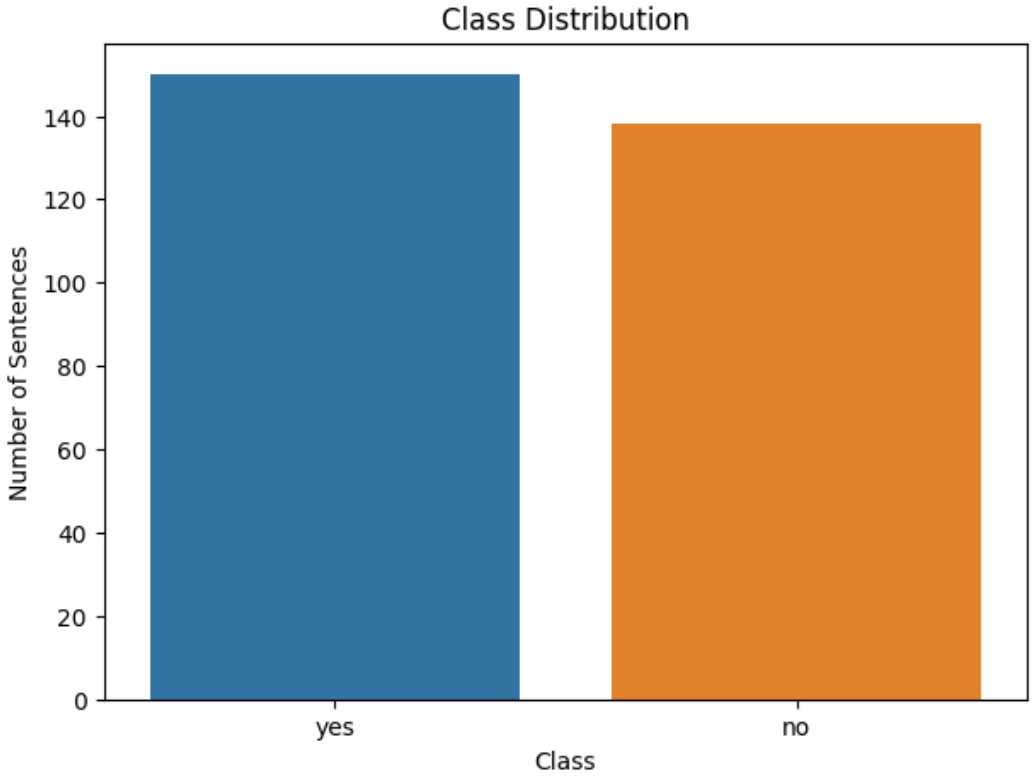


Figure 9 Class Distribution Bar Chart

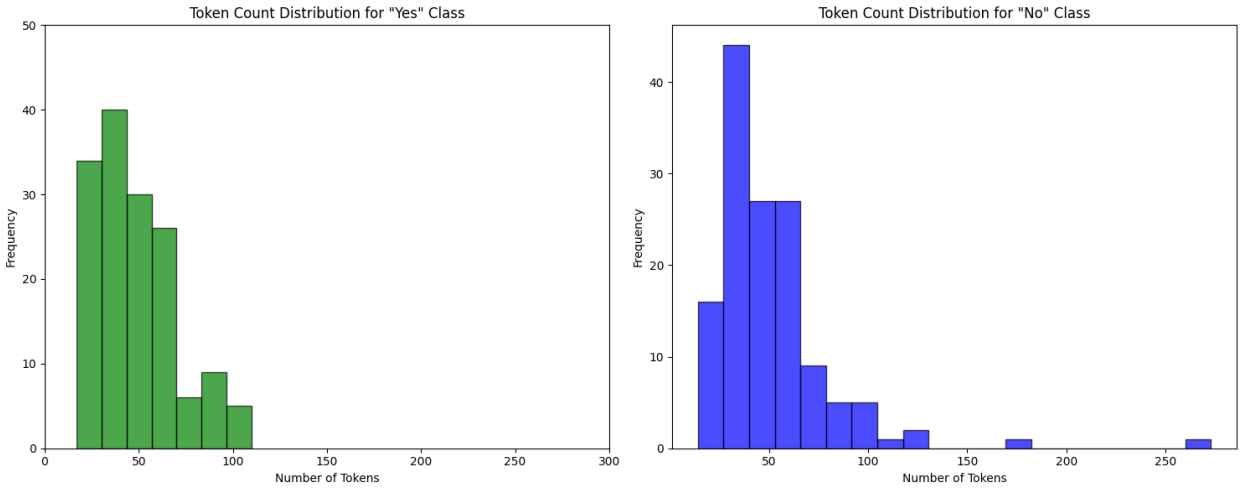


Figure 10 Token Count Distribution Histograms

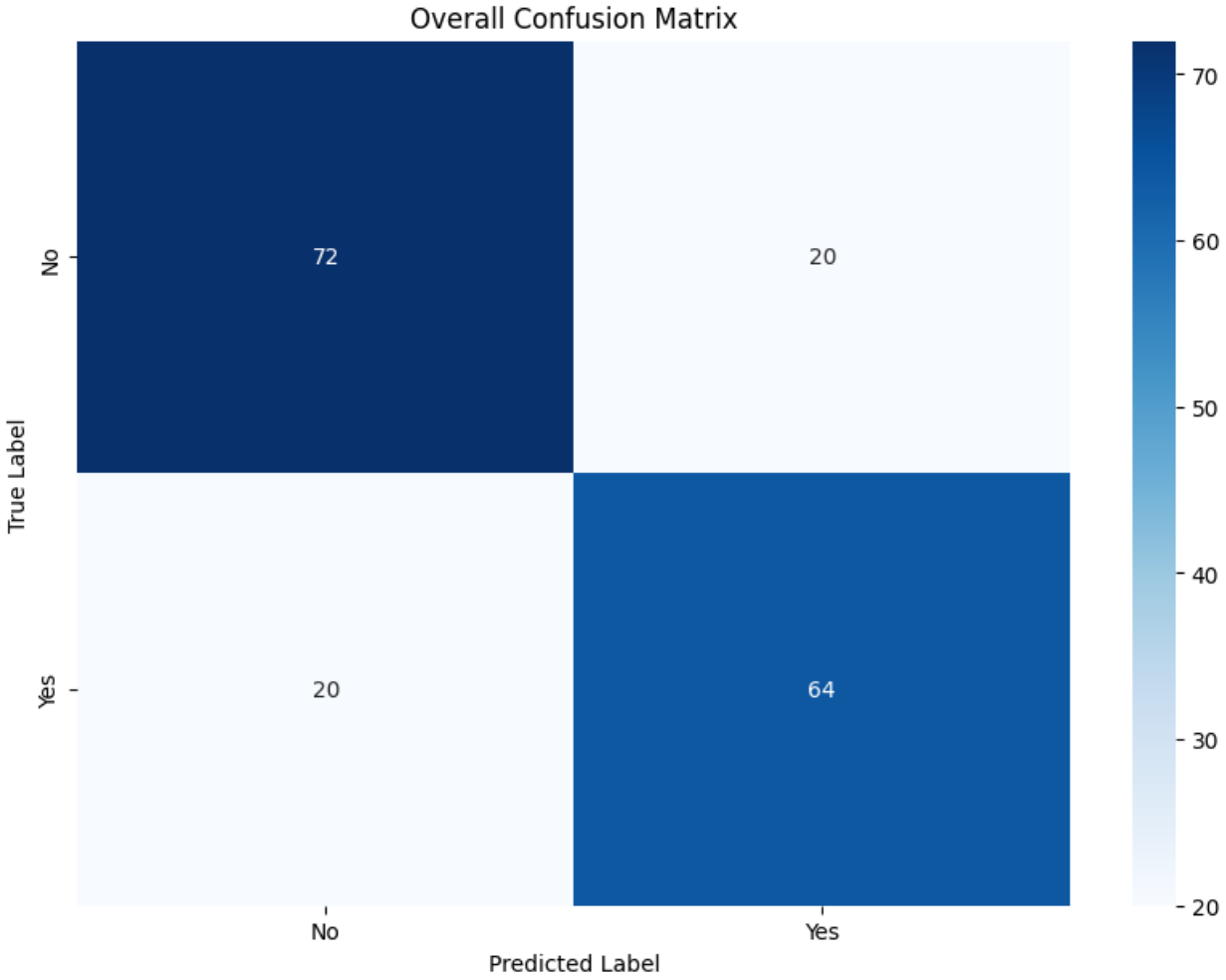


Figure 11 Overall Confusion Matrix