Sentiment Analysis Model and Usefulness Score Model for Netflix Reviews Based on Machine Learning and Deep Learning Techniques

Cardiff University School of Computer Science and Informatics

BSc Computing

Tianfeng GE - 22011387

Supervisor by Luis Espinosa-Anke

6th September 2024

Abstract

This paper uses the Netflix review dataset as the basis for an in-depth study of the two key tasks of sentiment analysis and usefulness scoring. TF-IDF and BERT are used as feature extraction methods, and logistic regression, random forest, XGBoost and LSTM models are constructed. The models are tuned using multiple methods to achieve sentiment analysis and usefulness scoring prediction of Netflix review data. For these two tasks, this paper comprehensively evaluates and compares the performance of the first three machine learning models using evaluation metrics such as accuracy, precision, recall, F1 score and confusion matrix. For the LSTM deep learning model, this paper comprehensively evaluates the model's performance and behavior through a classification report, overall accuracy and visualization of the loss and accuracy curves during model training. The research results provide new insights into user feedback analysis and demonstrate the advantages and limitations of different models in practical applications.

Acknowledgements

I am deeply grateful to my advisor, Luis Espinosa-Anke, for his help in this research. He not only provided valuable ideas for the two models on the dataset, but also helped me complete the initial schedule, which enabled me to advance the research in an orderly manner according to the plan. In addition, he also suggested that I build things like GitHub repositories, which further improved the organization and reproducibility of the research. In particular, he provided me with important guidance and support during the model building and report writing process. I would like to express my sincere gratitude for his help, professional advice and patient guidance.

In addition, I would like to thank Kaggle for providing data support, which enabled me to obtain and use high-quality datasets for research. At the same time, I am also grateful to the library search of Cardiff University for providing paper reference support, which laid a solid foundation for my literature research and theoretical foundation.

Finally, I would like to thank everyone who helped and supported me during this process.

Table of Contents

List of Figures

1. Introductory

1.1 Background and motivation

“User-generated content (UGC) has evolved from a mere web phenomenon into a significant component of the digital world.” (Tomaiuolo, 2012). With the popularity of social media and online platforms, UGC has become increasingly important in the digital world. As the world's leading streaming platform, Netflix has many user reviews, which not only influence other users' viewing decisions, but also provide valuable information for content creators and platform optimization. Tomaiuolo (2012) argues that “as UGC continues to grow in volume and complexity, data analytics has become an important tool for understanding user needs and behaviour.” In this context, sentiment analysis and usefulness scoring models become key tools for understanding user sentiment, behavior and content preferences.

However, due to the diversity and complexity of UGC, these tasks face challenges such as data sparsity, high dimensions, and diverse changes in sentiment, which require the use of a variety of advanced feature extraction methods, machine learning and deep learning models to solve. TF-IDF is a simple but effective text feature

extraction method that measures the importance of words in a document by calculating the term frequency and inverse document frequency. Although TF-IDF cannot capture deep semantic information, it is simple to calculate and fast, making it particularly suitable for traditional machine learning models (such as logistic regression). TF-IDF can be used as a baseline method to compare with more complex BERT models, helping us understand the differences in performance of different methods in the sentiment analysis task. Compared with traditional feature extraction methods (TF-IDF), BERT is a pre-trained language model based on a bidirectional Transformer architecture that can understand context from both left to right and right to left, capturing richer semantic information in sentences. BERT can identify more complex emotions and context dependencies, and it excels at processing long texts and polysemy. Therefore, in the task of sentiment analysis, BERT's bidirectional and context-sensitive characteristics can better identify the implicit sentiment and contextual information in the text, improving the accuracy and robustness of the model.

When it comes to machine learning models, it is inevitable to mention the most traditional machine learning model – logistic regression. As a simple and easy-to-interpret linear model, logistic

regression performs well in cases where the number of features is small, or the features are linearly separable. Its interpretability allows us to directly understand the contribution of each feature. In contrast, random forests, an ensemble learning method that combines multiple decision tree models, exhibit good robustness and the ability to handle non-linear relationships, making them particularly suitable for sentiment analysis tasks with high data dimensions and noise. XGBoost, on the other hand, is an improved version of gradient-boosted trees. XGBoost is known for its efficient parallel computing and accuracy and is suitable for large datasets and complex pattern recognition tasks.

In deep learning models, an LSTM is a type of recurrent neural network suitable for processing sequential data (such as text). By introducing ‘memory’ units, it captures long-term dependencies in sequential data. The advantage of LSTM in sentiment analysis tasks is that it can learn long- and short-distance emotional clues in text sequences and retain these clues to improve classification performance. It is especially suitable for complex sentiment analysis tasks that require the capture of contextual relationships.

In this study, TF-IDF provides a simple and quick baseline

comparison, while BERT can better understand complex contextual information. The diverse selection of models such as LSTM, logistic regression, random forest and XGBoost helps to comprehensively evaluate the performance of the models in the sentiment analysis and usefulness scoring tasks. By combining these methods, I expect to be able to more accurately capture the emotional tendencies of user comments, provide more reliable usefulness scores, and support the content optimization and recommendation systems of the Netflix platform.

1.2 Research Target

The main objectives of this study are as follows:

1. **Sentiment analysis model:**

The model predicts the sentiment of user reviews by training and testing on a dataset split by date based on time series using review data that does not overlap in time. By accurately identifying and analyzing user sentiment, this model helps the Netflix platform better understand user viewing emotions and preferences, thereby improving the recommendation system and content optimization strategies.

2. Usefulness scoring model：

By modelling and training the number of ‘thumbs up’ for user comments, a usefulness scoring model is constructed to assess the

degree to which comments are helpful to other users. This model will identify the most valuable user feedback for the platform, improve the overall user experience, and optimize the content recommendation mechanism.

1.3 Research Contribution

This study comprehensively evaluates the performance of two feature extraction methods (TF-IDF and BERT) and multiple machine learning and deep learning models (logistic regression, random forest, XGBoost, and LSTM) in sentiment analysis and usefulness scoring tasks. This paper systematically compares the performance differences between the traditional TF-IDF model and the advanced BERT model. By using multiple models such as logistic regression, random forest, XGBoost and LSTM, this paper reveals the advantages and limitations of various methods under different data conditions, providing an important reference for selecting appropriate sentiment analysis and usefulness scoring methods. This paper also uses multiple evaluation metrics (such as accuracy, precision, recall and F1 score) and combines them with data visualization techniques to comprehensively analyze the prediction performance of each model. These results not only help to understand the performance of different models on Netflix user review data, but also provide data support for the subsequent

improvement and optimization of sentiment analysis and usefulness scoring algorithms.

The results of this study can more accurately capture the sentiment orientation and usefulness scoring of user reviews, providing strong support for content optimization, user experience improvement, and the improvement of personalized recommendation systems on the Netflix platform. At the same time, the methods and results of this study can also provide reference and reference for UGC analysis in other fields.

2. Related Work

2.1 Sentiment Analysis

Sentiment analysis is an important task in natural language processing (NLP) that aims to extract subjective information from text. *"Machine-learning-based techniques proposed for sentiment analysis problems can be divided into two groups: (1) traditional models and (2) deep learning models. " (Dang, Moreno-García and De la Prieta, 2020).* Traditional methods such as dictionary-based methods and machine learning methods (such as support vector machines and Bayesian networks) have been widely used. In recent years, deep learning techniques (such as LSTM and BERT) have made significant progress due to their powerful contextual modelling capabilities.

2.2 Usefulness Score

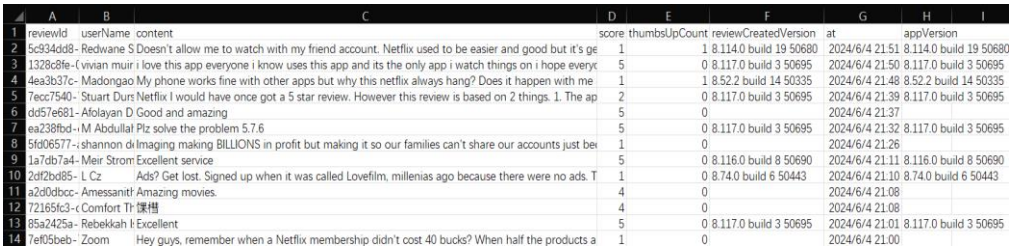
The usefulness score is generally used to predict the value of user reviews. Traditional feature engineering methods and machine learning models (such as logistic regression and random forests) have performed well in this area. In recent years, the introduction of deep learning models has further improved prediction performance, especially in terms of modelling complex text features.

3. Methodology

3.1 Dataset

This study used the Netflix review dataset from Kaggle. The dataset records the review information submitted by Netflix users on the Google Play Store, including the following fields:

- *reviewid*: Unique comment identifier to uniquely identify each comment
- userName: Google Play Store username of the user who commented
- content: Reviews, the text of user-specific reviews for Netflix
- score: Ratings, User Ratings for Netflix
- thumbsUpCount: Likes, the number of 'likes' on the comment by other users
- at: Comment timestamp, date and time of comment submission
- reviewCreatedVersion: App version at the time of the review, the version number of the Netflix app at the time the user submitted the review
- appVersion: App Version, the current Netflix app version number that distinguishes between reviews and app versions.



	A	B	C	D	E	F	G	H	I
1	reviewId	userName	content	score	thumbsUpCount	reviewCreatedVersion	at	appVersion	
2	5c934dd8-	Redwane S	Doesn't allow me to watch with my friend account. Netflix used to be easier and good but it's ge	1	1	8.114.0 build 19 50680	2024/6/4 21:51	8.114.0 build 19 50680	
3	1328c8fe-(	vivian muir	i love this app everyone i know uses this app and its the only app i watch things on i hope everyc	5	0	8.117.0 build 3 50695	2024/6/4 21:50	8.117.0 build 3 50695	
4	4ea3b37c-	Madongao	My phone works fine with other apps but why this netflix always hang? Does it happen with me	1	1	8.52.2 build 14 50335	2024/6/4 21:48	8.52.2 build 14 50335	
5	7ecc7540-	Stuart Durs	Netflix I would have once got a 5 star review. However this review is based on 2 things. 1. The ap	2	0	8.117.0 build 3 50695	2024/6/4 21:39	8.117.0 build 3 50695	
6	dd57e681-	Afolayan D	Good and amazing	5	0		2024/6/4 21:37		
7	ea238fbd-(	M Abdullal	Plz solve the problem 5.7.6	5	0	8.117.0 build 3 50695	2024/6/4 21:32	8.117.0 build 3 50695	
8	5fd06577-(	shannon di	Imaging making BILLIONS in profit but making it so our families can't share our accounts just bec	1	0		2024/6/4 21:26		
9	1a7db7a4-	Meir Strom	Excellent service	5	0	8.116.0 build 8 50690	2024/6/4 21:11	8.116.0 build 8 50690	
10	2df2bd85-	L Cz	Ads? Get lost. Signed up when it was called Lovefilm, millenias ago because there were no ads. T	1	0	8.74.0 build 6 50443	2024/6/4 21:10	8.74.0 build 6 50443	
11	a2d0dbcc-	Amessanit	Amazing movies.	4	0		2024/6/4 21:08		
12	72165fc3-(	Comfort Th	谋概	4	0		2024/6/4 21:08		
13	85a2425a-	Rebekkah I	Excellent	5	0	8.117.0 build 3 50695	2024/6/4 21:01	8.117.0 build 3 50695	
14	7ef05beb-	Zoom	Hey guys, remember when a Netflix membership didn't cost 40 bucks? When half the products a	1	0		2024/6/4 21:00		

P 1.Examples of dataset

3.2 Sentiment Analysis Model

3.2.1 Analysis Data

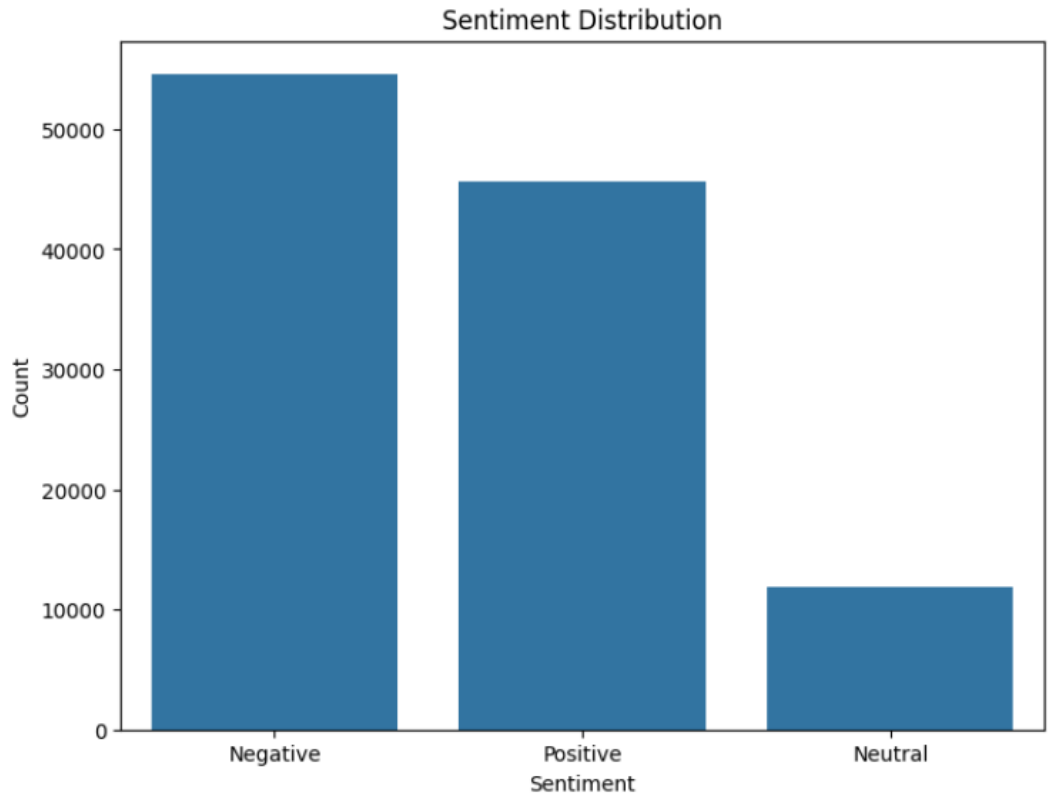
”Data analysis is often the first step in any data science project. It involves exploring and understanding the data to uncover insights and patterns.” (Shan, Wang, Chen, & Song, 2016). Data analysis is a necessary step to build an efficient sentiment analysis model. First, a basic inspection of the dataset is performed, including looking at the first few rows of data to understand the data structure and check for missing values, to ensure the quality and integrity of the data and lay a solid foundation for subsequent model training. Next, a sentiment column is created based on the score column, and the score values are converted into ‘positive’, ‘neutral’ or ‘negative’ sentiment labels. The creation of sentiment labels converts quantitative scores into sentiment categories, enabling the model to perform effective classification tasks. The distribution of sentiment labels is counted and visualized to understand the number and distribution of reviews in different sentiment categories. This helps identify imbalances in the data and guides data preprocessing and

model optimization strategies.

```
                               reviewId            userName  \
0  5c934dd8-6417-4653-aa21-2d093f468d72      Redwane Stayka
1  1328c8fe-0596-41c6-9ddf-20a5163a5c50         vivian muir
2  4ea3b37c-0478-4eb8-9754-7a27f917100c  Madongaolou Edward
3  7ecc7540-7968-4ba2-be6c-bdf3b0311daf      Stuart Durston
4  dd57e681-5ae5-4b41-909d-8a868fde751a   Afolayan Damilola

                                        content  score  thumbsUpCount  \
0  Doesn't allow me to watch with my friend accou...      1              1
1  i love this app everyone i know uses this app ...      5              0
2  My phone works fine with other apps but why th...      1              1
3  Netflix I would have once got a 5 star review....      2              0
4                                Good and amazing      5              0

     reviewCreatedVersion                   at             appVersion
0  8.114.0 build 19 50680  2024-06-04 21:51:39  8.114.0 build 19 50680
1   8.117.0 build 3 50695  2024-06-04 21:50:37   8.117.0 build 3 50695
2   8.52.2 build 14 50335  2024-06-04 21:48:50   8.52.2 build 14 50335
3   8.117.0 build 3 50695  2024-06-04 21:39:35   8.117.0 build 3 50695
4                     NaN  2024-06-04 21:37:16                    NaN
reviewId                  0
userName                  1
content                   2
score                     0
thumbsUpCount             0
reviewCreatedVersion  16239
at                        0
appVersion            16239
dtype: int64

Distribution of sentiment labels:
sentiment
Negative    54613
Positive    45622
```

P 2.View and Analysis Data

```
Positive    45622
Neutral     11878
Name: count, dtype: int64
```

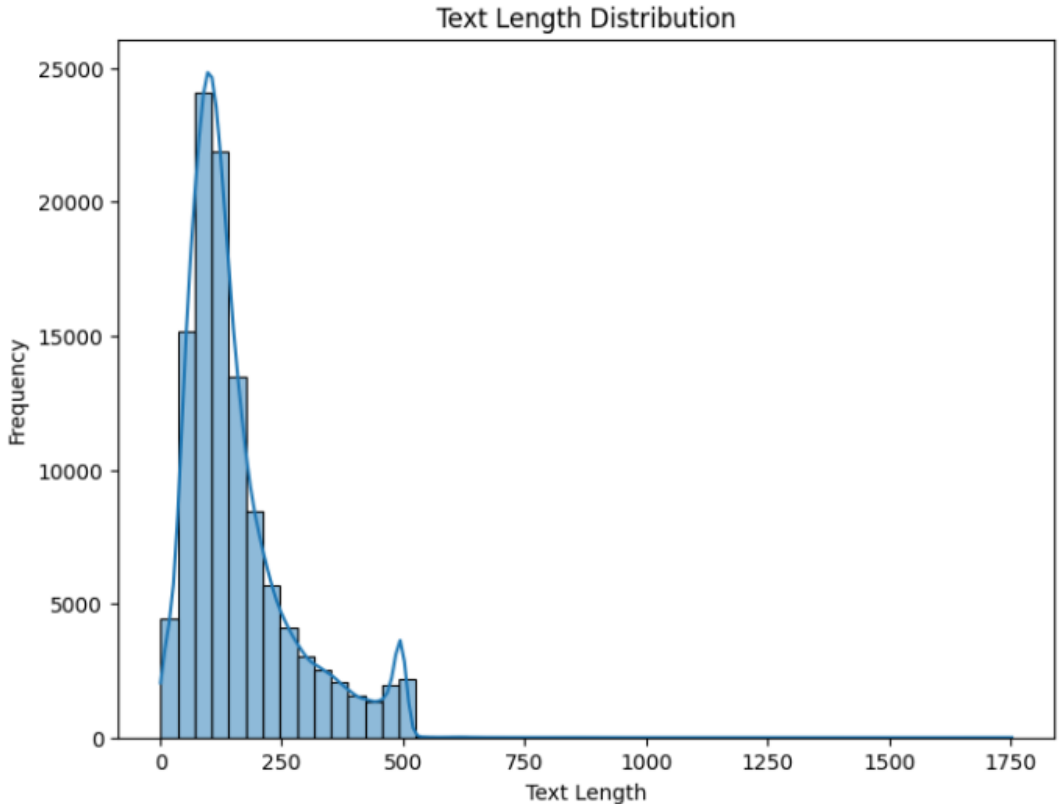


P 3.Sentiment Label Distribution Visualization in sentiment analysis model

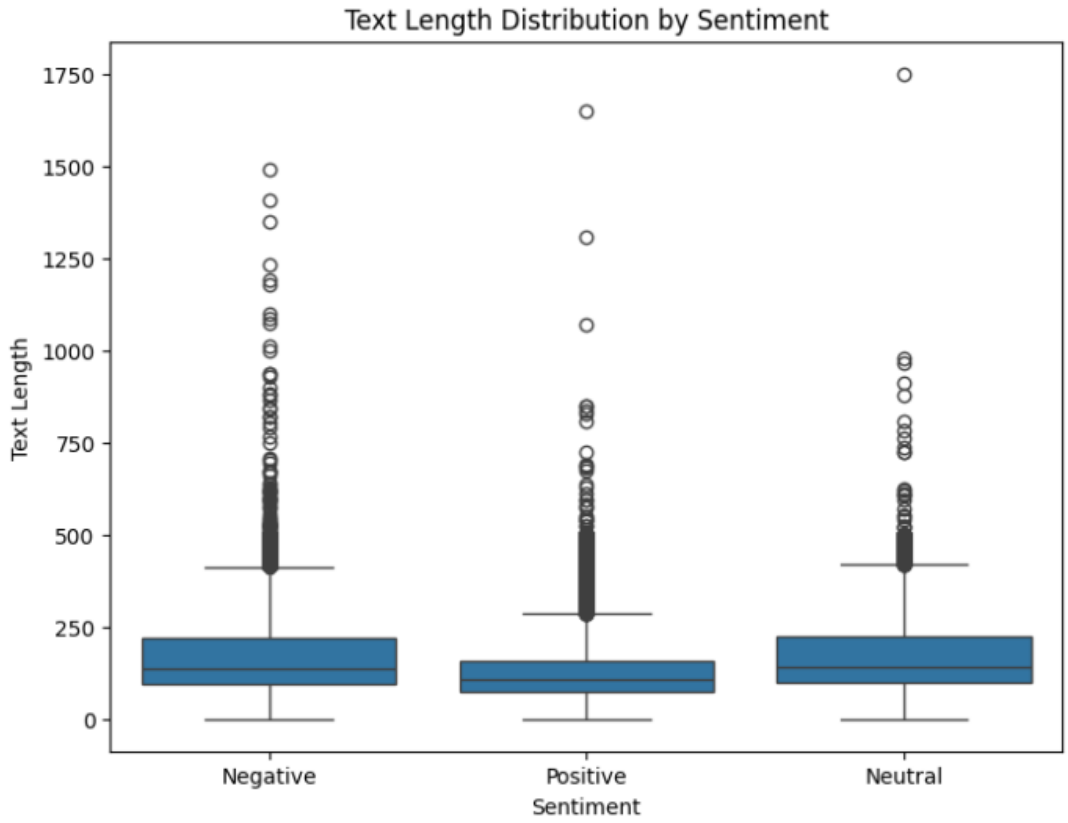
Subsequently, I first conducted a detailed analysis of the distribution of the text length of each comment, including calculating descriptive statistics of the text length and visualizing them with histograms and kernel density estimation (KDE) plots. Histograms show the frequency of different text length intervals, while KDE plots provide a smooth curve of the text length distribution. These charts provide a visual understanding of the common intervals and trends in text length. This analysis reveals the characteristics of the comments, providing a basis for the selection of model input features and the

formulation of text processing strategies.

```
Descriptive statistics of text length:
count    112113.000000
mean        158.328695
std         111.709309
min           1.000000
25%          85.000000
50%         124.000000
75%         196.000000
max        1752.000000
Name: text_length, dtype: float64
```



P 4.Histogram and KDE of the text length distribution for the sentiment analysis model



P 5.Box plot of the distribution of text length for the sentiment analysis model

```
# View data
# View the first few rows of data
print(df.head())
# Check for missing values
print(df.isnull().sum())
# Create a sentiment label column based on the score column 创建情感标签列，基于score列
def label_sentiment(score):
    if score >= 4:
        return 'Positive'
    elif score == 3:
        return 'Neutral'
    else:
        return 'Negative'
df['sentiment'] = df['score'].apply(label_sentiment)

# Check the distribution of sentiment labels 检查情感标签的分布
print("\nDistribution of sentiment labels: ")
print(df['sentiment'].value_counts())

# Visualizing the distribution of sentiment labels 可视化情感标签的分布
plt.figure(figsize=(8, 6))
sns.countplot(x='sentiment', data=df)
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Analyze text length distribution 分析文本长度分布
df['text_length'] = df['content'].apply(lambda x: len(str(x)))
print("\nDescriptive statistics of text length:")
print(df['text_length'].describe())

# Visualizing text length distribution 可视化文本长度分布
plt.figure(figsize=(8, 6))
sns.histplot(df['text_length'], bins=50, kde=True)
plt.title('Text Length Distribution')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()

# Text length distribution under each sentiment label 各情感标签下的文本长度分布
plt.figure(figsize=(8, 6))
sns.boxplot(x='sentiment', y='text_length', data=df)
plt.title('Text Length Distribution by Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Text Length')
plt.show()
```

P 6.Code for the analysis data of the sentiment analysis model

3.2.2 Data preprocessing

"It is essential because the quality of data preparation directly impacts the effectiveness and accuracy of the analytics." (Svolba, 2006). Data preprocessing is a crucial step in building a sentiment analysis model, and its purpose is to ensure the quality and consistency of the data. First, this paper removes rows with missing

values in the ‘username’ and ‘content’ columns, and removes the ‘reviewCreatedVersion’ and ‘appVersion’ columns, which are not related to the sentiment analysis task. Next is text cleaning, which involves removing HTML tags and non-alphabetic characters, and converting the text to lowercase. To ensure the linguistic uniformity of the data, this paper only retains reviews containing pure English characters and removes all blank text. Subsequently, word segmentation was performed, and common stop words were removed to reduce noise.

After cleaning and preprocessing the data, this paper examines the data and verifies its completeness and accuracy by outputting missing value statistics, data preview, and data structure information. These steps ensure data consistency and lay a solid foundation for the subsequent training of the sentiment analysis model.

```
reviewId          0
userName          0
content           0
score             0
thumbsUpCount     0
at                0
sentiment         0
text_length       0
cleaned_content   0
tokens            0
dtype: int64
                                    reviewId       userName  \
21675  e940dd6f-b642-44a6-8b2e-48e69d511a8e  A Google user
55124  b9ca4a19-91ec-4c81-9240-5dd84408a7d7  A Google user
33555  a9c91494-6787-4584-95d6-80f7c52e1d9a  A Google user
17928  32e36441-44c3-465d-9161-8a2d617d685b  A Google user
53961  7aff0740-fd01-4191-9271-f689820740a5  A Google user

                                                 content  score  \
21675  There are still some fantastic Netflix origina...      4
55124  Dont really use the app on my phone as much as...      4
33555  I love Netflix but on my LG v30+ all HDR looks...      3
17928  Well I downloaded it recently on my new phone ...      1
53961  Used to work well, now doesn't recognize my pa...      3

       thumbsUpCount                  at sentiment  text_length  \
21675              0 2018-09-12 07:22:12  Positive          260
55124              0 2018-09-12 07:33:31  Positive          112
33555              0 2018-09-12 07:38:21   Neutral          175
17928            124 2018-09-12 07:43:26  Negative          315
53961              3 2018-09-12 08:07:05   Neutral          105

                                       cleaned_content  \
21675  there are still some fantastic netflix origina...
55124  dont really use the app on my phone as much as...
33555  i love netflix but on my lg v all hdr looks ye...
17928  well i downloaded it recently on my new phone ...
53961  used to work well now doesnt recognize my pass...
```

P 7.Data structure after processing

The time column `at` is converted to a datetime format and sorted. Next, this paper analyses the time range of the ‘at’ column and calculates monthly and yearly statistics of the number of reviews to understand the temporal distribution of the data. Based on these temporal statistics, this paper selects ‘2024-01-01’ as the split point of the dataset and divides the data into training and test sets by date. The training set contains data before this date, while the test set

contains data after this date. Finally, this article extracted text labels and sentiment labels as training data and test data respectively and checked the size of the dataset and the amount of data before and after the split point. These steps ensure the comprehensiveness of the data and the reasonable distribution of the time series, laying a solid foundation for the training and evaluation of the model.

```
Time Scale:
Earliest Date: 2018-09-12 07:22:12
Latest Date: 2024-06-04 21:51:39

Volume of statistics by month:
at
2018-09     702
2018-10    1569
2018-11    1376
2018-12    1155
2019-01    1188
           ...
2024-02    2927
2024-03    3231
2024-04    2743
2024-05    4616
2024-06     627
Freq: M, Name: count, Length: 70, dtype: int64

Volume of statistics by year:
at
2018     4802
2019    16879
2020    28569
2021    17736
2022    15122
2023    12850
2024    16028
Freq: Y-DEC, Name: count, dtype: int64
```

P 8.Time distribution statistics

```
6]: # Data processing
    # Remove rows with missing values for `userName` and `content`
    df = df.dropna(subset=['userName', 'content'])

    # Remove `reviewCreatedVersion` and `appVersion` columns
    df = df.drop(columns=['reviewCreatedVersion', 'appVersion'])

    # Text cleaning functions
    def clean_text(text):
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        text = text.lower()
        return text

    # Function to check whether it contains non-English characters
    def is_english(text):
        return re.fullmatch(r'[a-zA-Z\s]*', text) is not None

    # Functions for word segmentation and stop word removal
    def tokenize_and_remove_stopwords(text):
        tokens = word_tokenize(text)
        stop_words = set(stopwords.words('english'))
        filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
        return filtered_tokens

    df['cleaned_content'] = df['content'].apply(clean_text)
    df = df[df['cleaned_content'].apply(is_english)]
    df = df[df['cleaned_content'].str.strip() != '']
    df['tokens'] = df['cleaned_content'].apply(tokenize_and_remove_stopwords)

    # Convert the `at` column to `datetime` format
    df['at'] = pd.to_datetime(df['at'])

    # Sort data by date
    df = df.sort_values(by='at')

    df.to_csv('Sentiment Analysis Model _ TF-IDF.csv', index=False)
```

P 9.Data preprocessing code for sentiment analysis model

3.2.3 Feature Extraction

TF-IDF (Term Frequency-Inverse Document Frequency):

“In automatic text retrieval systems, the TF-IDF term weighting scheme has been found to significantly improve the retrieval performance by effectively distinguishing between relevant and non-relevant documents.”(Salton & Buckley, 1988). TF-IDF is a statistical method commonly used in text analysis and information retrieval to assess the importance of words in a collection of documents. In sentiment analysis, TF-IDF is widely used to extract text features as feature vectors to help models better understand and classify the

sentiment in the text. By filtering out common meaningless words (such as ‘and’ and ‘is’, etc.), TF-IDF provides an effective way to identify and highlight important words that are more representative of the content of the document, laying a solid foundation for model training and classification. The formula for calculating it is:

$$\mathrm{TF}-\mathrm{IDF}(\mathrm{t},\mathrm{d})=\mathrm{TF}(\mathrm{t},\mathrm{d})\times\mathrm{IDF}(\mathrm{t},\mathrm{D})$$

- TF (Term Frequency): TF measures how often a word occurs in a document. TF is the number of times the word occurs in the document divided by the total number of words in the document. The formula is:

$$\mathrm{TF}(\mathrm{t},\mathrm{d})=\frac{\boldsymbol{Number\ of\ times\ term\ t\ appears\ in\ document\ d}}{Total\ number\ of\ terms\ in\ document\ d}$$

- IDF (Inverse Document Frequency): The IDF measures the frequency of occurrence of a word in the entire document collection. The lower the IDF value, the more likely it is that the word occurs in most documents and the less informative it is. The higher the IDF value, the more likely it is that the word occurs in fewer documents and the more informative it is. The formula is:

$$IDF(t,D)=\log\left(\frac{Total\ number\ of\ documents\ in\ the\ corpus\ D}{Number\ of\ documents\ where\ term\ t\ appears}+1\right)$$

A high TF indicates that the word appears frequently in the current document. A high IDF indicates that the word is not common in the entire document set (i.e., it has high information value). A high TF-

IDF score means that the word appears frequently in this document and is relatively rare in other documents, so the word contributes more to the identification of the characteristics of this document.

```
In [9]: # Feature extraction: vectorization using TF-IDF 特征提取：使用TF-IDF向量化
        tfidf = TfidfVectorizer(max_features=5000)
        X_train_tfidf = tfidf.fit_transform(X_train)
        X_test_tfidf = tfidf.transform(X_test)
```

P 10.TF-IDF code for sentiment analysis model

BERT (Bidirectional Encoder Representation Model):

"BERT's bidirectional approach allows it to deeply understand both the left and right context of a word, making it more effective than previous models that only processed text in one direction. This results in significant improvements across a wide range of NLP tasks, including sentiment analysis, question answering, and text classification" (Devlin et al., 2019). BERT is a deep learning model based on the Transformer architecture and designed for natural language processing tasks. Unlike traditional word vector models, BERT captures contextual information through bidirectional encoding, which enables it to more accurately understand the semantics of sentences. When encoding words, BERT can consider the context on the left and right sides at the same time, which makes it particularly good at dealing with complex language phenomena. Through pretraining and fine-tuning strategies, BERT not only provides context-sensitive word embeddings, but also converts text

into high-dimensional context embedding vectors to better express the meaning of the text. These embedding vectors perform well in tasks such as sentiment analysis, enabling the model to fully understand and process the emotional tendencies of the text, greatly improving the accuracy of the analysis and the generalization ability of the model.

```python
# BERT Processing
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def embed_text(texts, tokenizer, model, max_length=256):
    inputs = tokenizer(texts, return_tensors='pt', padding=True, truncation=True, max_le
    with torch.no_grad():
        outputs = model(**inputs)
    cls_embeddings = outputs.last_hidden_state[:, 0, :].detach().cpu().numpy()
    return cls_embeddings

def batch_embed_texts(texts, batch_size=32):
    embeddings = []
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i + batch_size]
        batch_embeddings = embed_text(batch_texts, tokenizer, model)
        embeddings.append(batch_embeddings)
    return np.vstack(embeddings)

# Get BERT embedding vector
texts = df['cleaned_content'].tolist()
batch_size = 32

# Use multithreaded parallel processing    使用多线程并行处理
with ThreadPoolExecutor() as executor:
    bert_embeddings = np.vstack(list(executor.map(lambda x: embed_text([x], tokenizer, mo

# Add BERT embedding vectors to the dataframe  将BERT嵌入向量添加到数据框
df['bert_embeddings'] = list(bert_embeddings)

# Partition the dataset by date
split_date = '2024-01-01'
train_df = df[df['at'] < split_date]
test_df = df[df['at'] >= split_date]

# Extract BERT embedding vectors and sentiment labels 提取BERT嵌入向量和情感标签
X_train = np.vstack(train_df['bert_embeddings'].values)
y_train = train_df['sentiment'].values
X_test = np.vstack(test_df['bert_embeddings'].values)
y_test = test_df['sentiment'].values
```

P 11.Bert code for sentiment analysis model

3.2.4 Model

Four models were selected for sentiment analysis and usefulness

scoring: logistic regression, random forest, XGBoost and LSTM (long short-term memory network). These models represent linear models, ensemble learning models, gradient boosting models and deep learning models, respectively, and can perform effective classification tasks using different feature extraction methods (TF-IDF and BERT).

Logistic Regression：

"Logistic regression is a powerful tool for modeling binary outcomes, especially when the relationship between the independent variables and the dependent variable is not strictly linear" (Pampel, 2000). Logistic regression is a classic linear classification model that is widely used for binary classification tasks. The core idea is to fit the data by maximizing the log-likelihood function, thereby predicting the probability that a sample belongs to a certain class. Although the logistic regression model is relatively simple, it performs well when dealing with linearly separable datasets and is especially stable when dealing with high-dimensional sparse data (such as TF-IDF feature vectors). Its main advantages include ease of implementation, ease of interpretation and computational power. However, logistic regression can only capture linear relationships in the data and has difficulty dealing with complex non-linear patterns.

Logistic Regression on TF-IDF feature vectors

In this study, I first used a logistic regression model on the TF-IDF feature vector and used a variety of methods to adjust and optimize the model. First, I used a standard logistic regression model to perform a preliminary classification of the data. On this basis, I introduced category weight adjustment to address the problem of class imbalance, and to improve the classification performance of the ‘neutral’ category. In addition, I further optimized the model hyperparameters, such as the regularization parameter C and the penalty, through grid search (GridSearchCV) to find the best combination of model parameters. I tried different feature extraction and weight adjustment methods to improve the model's performance on high-dimensional sparse data (TF-IDF feature vectors). This series of adjustment steps ensured the stability and accuracy of the model when dealing with class imbalance and high-dimensional data, laying a solid foundation for the subsequent sentiment analysis task.

```python
# Logistic regression code summary:
# 1.Logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)


# 2.Adjust class weights to make the model focus more on neutral sentiment 调整类权重，使
model = LogisticRegression(max_iter=1000, class_weight={'Negative': 1, 'Neutral': 3, 'Posi
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)


# 3.Adjust class weights to make the model focus more on neutral sentiment 调整类权重，使
# neutral: 3 —>2
model = LogisticRegression(max_iter=1000, class_weight={'Negative': 1, 'Neutral': 2, 'Posi
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)


# 4.Hyperparameter Tuning (Grid Search)
# 超参数调优（网格搜索） 使用网格搜索来优化逻辑回归的正则化参数。

# Defining the parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

# Instantiating the Logistic Regression Model
model = LogisticRegression(max_iter=1000, class_weight='balanced')

# Grid Search
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='f1_weigl
grid_search.fit(X_train_tfidf, y_train)

# Output the best parameters
print("Best parameters:", grid_search.best_params_)

# Train the model using the best parameters
best_model = grid_search.best_estimator_
y_pred = model.predict(X_test_tfidf)
```

P 12. Logistic regression code for sentiment analysis model based on TF-IDF feature vector

```
# 5.Feature extraction optimization 特征提取优化
# Avoid affecting the original dataset
X_train2 = train_df['cleaned_content']
y_train2 = train_df['sentiment']
X_test2 = test_df['cleaned_content']
y_test2 = test_df['sentiment']

# Feature extraction: Use TF-IDF vectorization to extract single and double word features
tfidf = TfidfVectorizer(ngram_range=(1, 2), max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train2)
X_test_tfidf = tfidf.transform(X_test2)

# Logistic regression model
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train_tfidf, y_train2)
y_pred = model.predict(X_test_tfidf

# 6.Feature extraction optimization2 —> ngram_range=(1, 3)    特征提取优化2
# Use TF-IDF vectorization to extract single, double and triple word features使用TF-IDF向
tfidf = TfidfVectorizer(ngram_range=(1, 3), max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train2)
X_test_tfidf = tfidf.transform(X_test2)

# Logistic regression model
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train_tfidf, y_train2)
y_pred = model.predict(X_test_tfidf)
```

P 13.Logistic regression code for sentiment analysis model based on TF-IDF feature vector 2

Logistic Regression on BERT feature vectors

In this study, I use a BERT-based logistic regression model and feature vector for feature learning and prediction. First, I set the max_iter parameter to 1000 to ensure that the model is adequately trained and set class_weight='balanced' to handle class imbalance. After the training is complete, I make predictions on the test data and calculate the overall accuracy of the model. In addition, I generate a classification report that details the precision, recall, and F1 score for each sentiment category. These metrics provide detailed information about the model's performance on different sentiment labels and help me evaluate the model's ability to predict each category.

To more intuitively demonstrate the model's classification results, we have plotted a confusion matrix heatmap. The confusion matrix shows a comparison between the model's predictions and the actual labels in the form of a heatmap, clearly displaying the predicted and actual distributions for each category. In the heatmap, dark areas indicate that the model's predictions for these categories are more concentrated, while light areas reveal possible model confusions.

```
# Logistic Regression
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Overall Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("\nLogistic Regression Model Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Neutral', 'Positive']))

conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Neutral', 'Positive'
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

P 14.The code for the logistic regression of the affective analysis model based on the Bert feature vector

Random Forest：

“Random forests create a multitude of decision trees and use the majority vote to determine the final classification result, which significantly improves the model’s accuracy and robustness.”(Liaw, Wiener, 2002). Random forests are a type of ensemble learning method that performs classification or regression by constructing multiple decision trees and voting on their predictions. Its advantages are high accuracy and strong resistance to overfitting, so it performs well when dealing with high-dimensional data and

complex features. In addition, random forests can effectively handle missing values and imbalanced data. However, the disadvantages of random forests include high computational complexity, which may result in slow training and prediction, especially when dealing with large-scale datasets. At the same time, because the model is composed of many decision trees, its internal structure is less interpretable, making it difficult to understand the specific decision-making process of the model.

Random Forest on TF-IDF feature vectors

In this study, a collection learning method based on TF-IDF feature vectors is used to improve the performance of the sentiment analysis model, combining random forests with other classifiers.

First, I define three basic classifiers: logistic regression, random forests, and gradient boosting. These models are integrated by soft voting, with the voting result based on the predicted probability of each classifier. During training, we used TF-IDF feature vectors, and evaluated the ensemble model's overall accuracy, precision, recall and F1 score. The classification results were visually displayed using a confusion matrix. In this way, the ensemble model can combine the advantages of each classifier to overcome the disadvantages of a single model and improve classification results when dealing with complex high-dimensional data.

```
# 1.Ensemble Learning  集成学习
# Defining the Model
lr_model = LogisticRegression(max_iter=1000, class_weight='balanced')
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Model Ensemble Using Voting 使用投票法进行模型集成
voting_clf = VotingClassifier(estimators=[('lr', lr_model), ('rf', rf_model), ('gb', gb_mc
voting_clf.fit(X_train_tfidf, y_train)

# Model predictions
y_pred = voting_clf.predict(X_test_tfidf)

# Model Evaluation
print("Overall Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("\nEnsemble Learning report(Logistic regression, Random forests and Gradient boostin
print(classification_report(y_test, y_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=voting_clf.classe
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

P 15.Includes the Random Forest ensemble learning code for sentiment analysis models based on TF-IDF feature vectors

Random Forest on BERT feature vectors

In this study, I use a BERT feature vector-based random forest classifier for sentiment analysis. When building the model, I set n_estimators=100 to ensure that the random forest model can adequately learn complex patterns in the data and use class_weight='balanced' to handle class imbalance. After the model is trained, I make predictions on the test data and calculate the overall accuracy of the model. In addition, I generated a classification report that shows the precision, recall and F1 score for each sentiment category in detail. To visually display the model's classification results, I also plotted a confusion matrix heatmap.

```
# Random Forest
rf = RandomForestClassifier(n_estimators=100, class_weight='balanced')
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Overall Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("\nRandom Forest Model + Class Weight Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Neutral', 'Positive']))

conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Neutral', 'Positive'
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

P 16.Random Forest code for sentiment analysis model based on Bert feature vector

XGBoost：

XGBoost (Extreme Gradient Boosting) is a highly effective boosting algorithm that is widely used in classification and regression tasks. It improves the prediction accuracy of the model by gradually optimizing a series of decision trees and introduces a regularization mechanism to prevent overfitting and thereby improve the generalization ability of the model. Chen and Guestrin (2016) pointed out that "XGBoost is designed to handle the bias-variance tradeoff effectively by incorporating regularization, which enhances model generalization and performance". This regularization and optimization mechanism makes XGBoost particularly good at handling complex datasets. However, the training process for XGBoost can be complex and may be slow when resources are limited, but its advantages in terms of accuracy and flexibility make it the model of choice for practical applications.

XGBoost on TF-IDF feature vectors

In the process of further optimizing the model using the TF-IDF feature vector, I extended the ensemble learning method and added the XGBoost classifier (XGBClassifier) to form a four-model ensemble system including logistic regression, random forest, gradient boosting and XGBoost. XGBoost is a powerful enhancement method that can further improve classification performance through reinforcement learning. This extended model is also integrated through soft voting, which combines the predictions of different models.

```
# 2.More ensemble learning
# Defining more models
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

# Model Ensemble Using Voting
voting_clf = VotingClassifier(estimators=[('lr', lr_model), ('rf', rf_model), ('gb', gb_mc
voting_clf.fit(X_train_tfidf, y_train)

# Model predictions
y_pred = voting_clf.predict(X_test_tfidf)

# Model Evaluation
print("Overall Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("\nEnsemble Learning report2(Random forests, Gradient boosting and XGBoost):")
print(classification_report(y_test, y_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=voting_clf.classe
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

P 17.Includes XGBoost ensemble learning code for sentiment analysis models based on TF-IDF feature vectors

XGBoost on BERT feature vectors

In this study, I used an XGBoost model based on BERT feature

vectors for sentiment analysis. First, I configured the parameters of the XGBoost classifier, where objective='multi:softmax' indicates a multi-class classification task, num_class=3 indicates that there are three sentiment categories, eval_metric='mlogloss' is used to evaluate the model's multi-class log loss, and use_label_encoder=False avoids warnings from the label encoder. After the model was trained, I made predictions on the test set and calculated the overall accuracy of the model. I also generated the classification report for the XGBoost model and plotted a heatmap of the confusion matrix.

```
# xgboost

xgb_model = xgb.XGBClassifier(objective='multi:softmax', num_class=3, eval_metric='mlogloss', use_label_en

xgb_model.fit(X_train, y_train)

y_pred_xgb = xgb_model.predict(X_test)

print("Overall Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred_xgb) * 100))

print("\nXGBoost Model Report:")
print(classification_report(y_test, y_pred_xgb, target_names=['Negative', 'Neutral', 'Positive']))


conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Neutral', 'Posit
plt.title('Confusion Matrix - XGBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

P 18.XGBoost code for sentiment analysis based on the Bert feature vector

LSTM:

LSTM (Long Short-Term Memory) is a special type of recurrent neural network (RNN) that is specifically designed to process and predict long-term dependencies in time series data. LSTM excels at processing sequential data by introducing a gating mechanism that

can effectively capture and retain long-term contextual information. Compared with traditional RNNs, LSTM solves the problems of gradient disappearance and gradient explosion through a combination of forget gates, input gates and output gates, which greatly improves the training stability and performance of the model. As stated by Hochreiter and Schmidhuber (1997), "LSTM networks are able to learn long-term dependencies in sequences due to their ability to maintain a cell state over time, which mitigates the vanishing gradient problem" (Hochreiter & Schmidhuber, 1997). Although LSTM is very powerful at capturing complex patterns in time series data, its training process can be time-consuming, and the model complexity is high. Nevertheless, LSTM still performs well in tasks such as natural language processing and time series prediction and is an effective tool for processing complex sequence data.

LSTM on TF-IDF feature vectors

In this study, I used a variety of LSTM models based on TF-IDF variables to explore their performance in sentiment analysis, and optimized and improved them. First, I built a basic LSTM model that used an embedding layer to map words to dense vectors, then used an LSTM layer for feature extraction, and finally used a fully connected layer to output the classification result. This model

showed good basic performance after training. I then improved the model by increasing the dropout rate to reduce overfitting and further improving the model's expressiveness with a second LSTM layer. This improvement improved the model's performance during training and validation. I then introduced batch normalization to speed up training and stabilize the network's learning process. The model showed higher accuracy and a more stable training process when dealing with complex data. Finally, to address the class imbalance problem, I used a bidirectional LSTM model and introduced class weight and learning rate scheduling strategies to better handle the bias in the data. Each model recorded the changes in loss and accuracy during training and performed a detailed performance evaluation. Through these steps, I fully exploited the potential of the LSTM model in sentiment analysis and optimized its performance under various conditions.

```
#  LSTM Summary
# 1.LSTM model
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))  # Number of classes
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy
history = model.fit(X_train_pad, y_train, epochs=5, batch_size=32, validation_split=0.1, v

# 2.LSTM model with Dropout
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=128, input_length=max_len))
model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3, return_sequences=True))
model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy
history = model.fit(X_train_pad, y_train, epochs=5, batch_size=32, validation_split=0.1, v

# 3. LSTM model with Dropout and BatchNormalization
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=128, input_length=max_len))
model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3, return_sequences=True))
model.add(BatchNormalization())
model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy
history = model.fit(X_train_pad, y_train, epochs=5, batch_size=32, validation_split=0.1, v

# 4. Bidirectional LSTM model with class weights  具有类别权重的双向 LSTM 模型
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=128, input_length=max_len))
model.add(Bidirectional(LSTM(128, dropout=0.3, recurrent_dropout=0.3, return_sequences=Tru
model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy
# Set class weights
```

P 19.LSTM code for sentiment analysis model based on TF-IDF feature vector

LSTM on BERT feature vectors

In this study, I used an LSTM network to model sentiment data based on BERT feature vectors. First, I preprocessed the data by adding a time step dimension and reshaping the training and test data into a format suitable for LSTM input (i.e., (sample, time step = 1, feature)). When building the model, I designed a network with two LSTM layers. The first LSTM layer has 128 units, uses a dropout

rate of 0.3 and a recursive dropout rate of 0.3 to reduce the risk of overfitting, and is normalized via a batch normalization layer. The second LSTM layer has 64 units, also uses a dropout rate and recursive dropout rate, and immediately adds another batch normalization layer. I then used 32 ReLU activation function units in a fully connected layer (dense layer) and applied a dropout rate of 0.5. Finally, I handled the multiclass classification problem with an output layer with a SoftMax activation function with 3 units.

The model uses the Adam optimizer with a learning rate of 0.001 and the loss function is sparse_categorical_crossentropy. During training, I set 5 training cycles (epochs) and train with a batch size of 32, while reserving 10% of the data for validation. After training, we predict on the test data, calculate the overall accuracy of the model, and generate a classification report. To evaluate the model's performance during training, I plotted the training loss and validation loss curves, as well as the training accuracy and validation accuracy curves. These charts show the trend of the model's performance in each training cycle and help to visually understand the model's convergence and generalization capabilities.

```
# LSTM
# Reshape the data to add a time step dimension
X_train = np.expand_dims(X_train, axis=1)  # Reshape to (samples, timesteps=1, features)
X_test = np.expand_dims(X_test, axis=1)

model = Sequential()
model.add(LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), dropout=0.3, recurrent_dropout=0.3,
model.add(BatchNormalization())
model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.1, verbose=1)

y_pred_prob = model.predict(X_test)
y_pred_classes = y_pred_prob.argmax(axis=-1)

print("Overall Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred_classes) * 100))
print("\nLSTM Model + Dropout and BatchNormalization Report:")
print(classification_report(y_test, y_pred_classes, target_names=['Negative', 'Neutral', 'Positive']))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss over Epochs')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy over Epochs')
plt.legend()

plt.tight_layout()
plt.show()
```

P 20.LSTM code for sentiment analysis model based on Bert feature vector

3.3 Usefulness Score Model

3.3.1 Data preprocessing

Data preprocessing is crucial to ensuring the accuracy and validity of the usefulness scoring model. First, I deleted records with missing values in the username and content columns and deleted the comment creation version and application version columns, which were not relevant to the analysis. Next, I performed text cleaning to remove HTML tags and non-alphabetic characters and converted

the text to lowercase to unify the data format.

I further filtered out comments containing only English characters and removed blank text. For the text content, I performed word segmentation and stop word removal: the text was segmented into words, common stop words were removed, and words useful for analysis were retained. This processed text was stored in the Cleaned content column.

To simplify data processing, I retained the Cleaned content column and the thumbsUpCount column and removed the missing values in the thumbsUpCount column. When converting the thumbsUpCount numerical data into a categorical problem, I set a threshold: records with a thumbsUpCount greater than 0 are marked as 1 (meaning useful), otherwise they are marked as 0 (meaning useless). Finally, I saved the processed data as a CSV file for subsequent model training and analysis. These data preprocessing steps ensure the integrity and consistency of the data and provide a solid foundation for building a usefulness scoring model.

```
[5]: # Data processing
     # Remove rows with missing values for `userName` and `content`
     df = df.dropna(subset=['userName', 'content'])

     # Remove `reviewCreatedVersion` and `appVersion` columns
     df = df.drop(columns=['reviewCreatedVersion', 'appVersion'])

     # Text cleaning functions
     def clean_text(text):
         text = re.sub(r'<.*?>', '', text)
         text = re.sub(r'[^a-zA-Z\s]', '', text)
         text = text.lower()
         return text

     # Function to check whether it contains non-English characters
     def is_english(text):
         return re.fullmatch(r'[a-zA-Z\s]*', text) is not None

     # Functions for word segmentation and stop word removal
     def tokenize_and_remove_stopwords(text):
         tokens = word_tokenize(text)
         stop_words = set(stopwords.words('english'))
         filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
         return filtered_tokens

     df['cleaned_content'] = df['content'].apply(clean_text)
     df = df[df['cleaned_content'].apply(is_english)]
     df = df[df['cleaned_content'].str.strip() != '']
     df['tokens'] = df['cleaned_content'].apply(tokenize_and_remove_stopwords)

     # Select the required columns: 'cleaned_content' and 'thumbsUpCount'
     df = df[['cleaned_content', 'thumbsUpCount']]

     # Remove rows where the value in the 'thumbsUpCount' column is NaN
     df = df.dropna(subset=['thumbsUpCount'])

     # Since 'thumbsUpCount' is a numerical data, it needs to be converted into a classificatio
     # If thumbsUpCount > 0, mark it as 1 (useful), otherwise it is 0 (useless)
     df['thumbsUpCount'] = df['thumbsUpCount'].apply(lambda x: 1 if x > 0 else 0)

     df.to_csv('Usefulness Score Model_TF-IDF.csv', index=False)
```

P 21.Data preprocessing code for Usefulness Score model

3.3.2 Feature Extraction

The usefulness score model uses the same feature extraction method as sentiment analysis. Here, I will not go into detail about the feature extraction method, but only explain how to use it:

TF-IDF:

First, I use TfidfVectorizer to convert the cleaned text data cleaned_content into TF-IDF feature vectors. To limit the feature

dimension, I set max_features=5000, which means that only the top 5000 most important feature words are retained. This step converts the text data into a numerical format suitable for machine learning models. Then, I separate the converted TF-IDF feature matrix X from the target variable thumbsUpCount, which represents the usefulness score. Next, I split the dataset into a training set and a test set, with 20% of the data used for testing and 80% for training the model. This process was completed using the train_test_split function, with the random seed random_state=42 set to ensure reproducible results. These feature extraction steps ensure the numerical processing of the data and provide a reliable basis for subsequent model training and evaluation.

```
6]: # Convert text data into numerical features
    vectorizer = TfidfVectorizer(max_features=5000)
    X = vectorizer.fit_transform(df['cleaned_content'])

    # Target variable
    y = df['thumbsUpCount']

    # Divide into training set and test set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

P 22.TF-IDF code for the Usefulness Score model

BERT:

First, I convert the target variable thumbsUpCount into a classification problem, where thumbsUpCount is marked as 1 if it is greater than 0 (useful) and 0 otherwise (not useful). Next, I load the pre-trained BERT model and its tokenizer (BertTokenizer and BertModel). The tokenizer is used to convert text data into an input

format acceptable to the BERT model, while the model is used to generate an embedded representation of the text. During the feature extraction process, I defined an embed_text function that uses the tokenizer to convert the text to the BERT input format and calculates the embedding representation of each text using the model. Specifically, I extract the hidden state of the [CLS] token output by the BERT model as the representation vector of each text. To process large-scale data, I implemented a batch_embed_texts function to process text data in batches to improve computational efficiency.

In practice, I batch convert text data into BERT embedding vectors. To speed up processing, I use multi-threaded parallel processing to batch input text data into the BERT model and collect the generated embedding vectors.

Finally, I separated the generated BERT feature vector bert_embeddings from the target variable thumbsUpCount and used train_test_split to divide the dataset into a training set and a test set, with 20% of the data used for testing and 80% for training. These steps ensure that the text data is effectively represented in the high-dimensional semantic space, laying a solid foundation for

subsequent model training and evaluation.

```
[1]: # BERT Processing

# Convert the target variable to a classification problem
df['thumbsUpCount'] = df['thumbsUpCount'].apply(lambda x: 1 if x > 0 else 0)

# Load the pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Define text vectorization function
def embed_text(texts, tokenizer, model, max_length=256):
    inputs = tokenizer(texts, return_tensors='pt', padding=True, truncation=True, max_length=max_length
    with torch.no_grad():
        outputs = model(**inputs)
    cls_embeddings = outputs.last_hidden_state[:, 0, :].detach().cpu().numpy()
    return cls_embeddings

# Batch processing function
def batch_embed_texts(texts, batch_size=16):
    embeddings = []
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i + batch_size]
        batch_embeddings = embed_text(batch_texts, tokenizer, model)
        embeddings.append(batch_embeddings)
    return np.vstack(embeddings)

# Batching comments into BERT vectors
texts = df['cleaned_content'].tolist()
batch_size = 32

# Use multithreaded parallel processing
with ThreadPoolExecutor() as executor:
    bert_embeddings = np.vstack(list(executor.map(lambda x: embed_text([x], tokenizer, model), texts)))

# Target variable
y = df['thumbsUpCount']

# Divide into training set and test set
X_train, X_test, y_train, y_test = train_test_split(bert_embeddings, y, test_size=0.2, random_state=42)
```

P 23.BERT code for the Usefulness Score model

3.3.3 Model

Model selection for the usefulness score model is like sentiment analysis. Here, I will not explain each model, but only how to use it:

Logistic Regression:

Logistic Regression on TF-IDF feature vectors

In this study, I used logistic regression based on TF-IDF feature vectors for feature learning and prediction. First, I built a basic logistic regression model and trained it using TF-IDF feature vectors.

After the model was trained, I made predictions on the test set and calculated the overall accuracy of the model. I also generated a classification report and confusion matrix to evaluate the model's performance. The confusion matrix shows the model's prediction results for each category, which helps me to further understand the model's accuracy and misclassification. In addition, I also used the trained model to calculate the usefulness score of each text in the dataset, which is the probability that the model predicts the text to be useful.

To improve the model's performance, I tried several improvement strategies. First, I used the class_weight='balanced' parameter to address the class imbalance problem and improve the model's ability to identify the minority class. Second, to address the model convergence problem, I increased the number of iterations of the logistic regression model to 200 and standardized the feature data to improve the stability and efficiency of the training process. Standardization helps speed up model convergence and improve performance.

Overall, these optimization steps have greatly improved the performance of the logistic regression model in the usefulness score

prediction task, providing a more reliable analytical tool for our text classification task.

```
#  Logistic Regression code Summary
# 1. Logistic Regression Model
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# 2.Logistic Regression Model + Class weight
# Use class_weight='balanced' to balance data in logistic regression
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)


# 3.Logistic Regression Model + Increase Iterations
# Solve the convergence problem: increase the number of iterations, data scaling 解决收敛问题:增加迭代次数,
# Standardize feature data   对特征数据进行标准化
scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Using a logistic regression model with increased max_iter  使用增加 max_iter 的逻辑回归模型
model = LogisticRegression(class_weight='balanced', max_iter=200, random_state=42)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test)

overall_accuracy = accuracy_score(y_test, y_pred) * 100
print("Overall Accuracy:  {:.2f}%".format(overall_accuracy))
print("Logistic Regression Model + Increase Iterations Report:")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Useful', 'Useful'], yticklabels=['No
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

df['usefulness_score'] = model.predict_proba(vectorizer.transform(df['cleaned_content']))[:, 1]
```

P 24.Logistic regression code based on the TF-IDF feature vector in the usefulness scoring model

Logistic Regression on BERT feature vectors

In this study, I also used a logistic regression model to classify texts based on BERT feature vectors. First, BERT was used to generate text feature vectors, and then these feature vectors were used to train a logistic regression model. The model was set to max_iter=200 to ensure sufficient training, and class_weight='balanced' was used to handle class imbalance. After

the training was complete, the model made predictions on the test set, and I calculated the overall accuracy and generated a classification report to evaluate the model's performance. The confusion matrix heatmap further shows the details of the prediction results, which helped me analyze the model's performance in practical applications.

```
# 1.Logistic Regression Model
model = LogisticRegression(max_iter=200, class_weight='balanced', random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)


overall_accuracy = accuracy_score(y_test, y_pred) * 100
print("Overall Accuracy:  {:.2f}%".format(overall_accuracy))
print("Logistic Regression Model Report:")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Useful', 'Useful'], yticklabels=['No
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

P 25.Logistic regression code based on BERT feature vector in Usefulness Score model

Random Forest:

Random Forest on TF-IDF feature vectors

In this study, I applied a random forest classifier based on TF-IDF feature vectors for feature learning and prediction. First, I trained a random forest model using TF-IDF feature vectors as input data. The model was fitted on the training set, and then predictions were made on the test set, and the overall accuracy of the model was calculated. The performance of the model was evaluated using a classification report and a confusion matrix, which shows the

prediction accuracy and recall rate for different categories.

To further optimize the model, I adjusted the class weights of the random forest in the second stage to address the class imbalance problem. I calculated the weight of each class and applied it to the model training, thereby improving the model's performance when dealing with imbalanced data. After training and evaluation, I also generated a classification report and confusion matrix to evaluate the performance of the adjusted model.

Finally, to improve the generalization ability of the model, I tuned the hyperparameters using GridSearchCV. First, the main hyperparameters of the random forest (such as n_estimators, max_depth, etc.) were determined through a preliminary grid search. Then, based on the preliminary results, the hyperparameter range was further refined and the minimum sample split parameter was optimized. Two rounds of grid search were used to find the optimal combination of hyperparameters, which was then used to train the final model. The performance of the final model on the test set was verified using the classification report and confusion matrix, and a usefulness score prediction result was generated. These steps ensured that the model had good predictive performance and

accuracy when processing real data.

```
# Random Forest code Summary
# 4.Random Forest Model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# 5.Random Forest Model + Class Weight
# Calculating Class Weights
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y), y=y)
class_weights_dict = {i: class_weights[i] for i in range(len(class_weights))}
model = RandomForestClassifier(class_weight=class_weights_dict, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# 6.Random Forest Model + Hyperparameter Tuning with GridSearchCV
# Hyperparameter Tuning: Use GridSearchCV to tune the hyperparameters of random forests超参数调优：使用Gri
rf = RandomForestClassifier(class_weight='balanced', random_state=42)

# First grid search 第一次网格搜索
param_grid_1 = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20]
}
grid_search_1 = GridSearchCV(estimator=rf, param_grid=param_grid_1, cv=5, scoring='f1', n_jobs=1)
grid_search_1.fit(X_train, y_train)
print(f"Best parameters (first stage): {grid_search_1.best_params_}")

# Further refine the search based on the results of the first stage 基于第一阶段的结果进一步细化搜索
best_params_stage_1 = grid_search_1.best_params_
param_grid_2 = {
    'min_samples_split': [2, 5, 10]
}

# Use the best parameters from the first stage 使用从第一阶段得到的最佳参数
rf = RandomForestClassifier(
    n_estimators=best_params_stage_1['n_estimators'],
    max_depth=best_params_stage_1['max_depth'],
    class_weight='balanced',
    random_state=42
)

grid_search_2 = GridSearchCV(estimator=rf, param_grid=param_grid_2, cv=5, scoring='f1', n_jobs=1)
grid_search_2.fit(X_train, y_train)

# Output the best parameters 输出最佳参数
print(f"Best parameters (second stage): {grid_search_2.best_params_}")
```

P 26.The random forest algorithm based on the TF-IDF feature vector in the utility scoring model

Random Forest on BERT feature vectors

In this study, I also applied a BERT-based random forest classifier for feature learning and prediction. First, I built a random forest classification model using the feature vectors generated by BERT as input data. To address the class imbalance problem, I calculated weights for each class and applied these weights to the model training. These weights help improve the model's performance when

dealing with data with an imbalanced class distribution.

```
# 2.Random Forest Model

class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y), y=y)
class_weights_dict = {i: class_weights[i] for i in range(len(class_weights))}

rf_model = RandomForestClassifier(class_weight=class_weights_dict, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

overall_accuracy_rf = accuracy_score(y_test, y_pred_rf) * 100
print("Overall Accuracy (Random Forest):  {:.2f}%".format(overall_accuracy_rf))
print("Random Forest Model Report:")
print(classification_report(y_test, y_pred_rf))

cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Useful', 'Useful'], yticklabels=[
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

P 27.Code for the random forest based on the BERT feature vector in the usefulness scoring model

XGBoost:

XGBOOST on TF-IDF feature vectors

In this study, I applied the XGBoost classifier based on the TF-IDF feature vector for feature learning and prediction. First, I built an XGBoost classifier and set the scale_pos_weight parameter to handle class imbalance. This parameter balances the classifier's sensitivity to different classes by adjusting the weights of positive and negative class samples. Once the model training was complete, I used the test set to make predictions and calculate the overall accuracy and classification report to show how the model performed on the different categories.

Next, I performed hyperparameter optimization to further improve the model performance. I first calculated the class weights and

defined a parameter grid, which included parameters such as max_depth, learning_rate, n_estimators and scale_pos_weight. Using GridSearchCV, I evaluated the effect of different parameter combinations during cross-validation and selected the best parameters. Finally, I used the optimized model to predict the original data, calculate the predicted probability, and add a usefulness score to the data frame. This process ensured that the XGBoost model could make full use of TF-IDF features to provide accurate text classification results and optimized the model's parameter settings to achieve the best performance.

```
# XGBoost code Summmary
# 7.Use ensemble learning model, XGBoost

# Divide the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
xgb_model = xgb.XGBClassifier(scale_pos_weight=class_weights_dict[1]/class_weights_dict[0], random_state=4
xgb_model.fit(X_train, y_train)


# 8.XGBoost Model + Tuning with GridSearchCV

# Calculating Class Weights
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y), y=y)
class_weights_dict = {i: class_weights[i] for i in range(len(class_weights))}

# Defining the parameter grid
param_grid = {
    'max_depth': [7],
    'learning_rate': [0.1],
    'n_estimators': [50],
    'scale_pos_weight': [1, class_weights_dict[1] / class_weights_dict[0]]
}

xgb_model = xgb.XGBClassifier(random_state=42)

# Hyperparameter optimization using GridSearchCV
grid_search = GridSearchCV(xgb_model, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Output the best parameters
print("Best Parameters:", grid_search.best_params_)
# Make predictions using the best parameters
best_xgb_model = grid_search.best_estimator_
y_pred = best_xgb_model.predict(X_test)

overall_accuracy = accuracy_score(y_test, y_pred) * 100
print("Overall Accuracy:  {:.2f}%".format(overall_accuracy))
print("XGBoost Model + Tuning with GridSearchCV Report:")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Useful', 'Useful'], yticklabels=['No
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
df['usefulness_score'] = model.predict_proba(vectorizer.transform(df['cleaned_content']))[:, 1]
```

P 28.XGBoost code based on the TF-IDF feature vector in the usefulness scoring model

XGBoost on BERT feature vectors

In this study, I also applied the XGBoost classifier based on BERT feature vectors for feature learning and prediction. First, I calculated the class weights to address the class imbalance in the dataset. We used the scale_pos_weight parameter to balance the weights of positive and negative class samples, which helped improve the model's performance on the minority class. Next, I built an XGBoost

classifier and applied the calculated class weights to the model. By training the model and making predictions on the test set, I obtained the overall accuracy and classification report. These results show the classification effect of the XGBoost model when processing BERT feature vectors. The visualization of the confusion matrix further helps analyse the model's performance on different categories. This approach combines BERT's powerful text representation capabilities with XGBoost's efficient classification performance, thereby improving the model's overall prediction ability.

```
# 3.XGBoost
class_weights = len(y_train) / (2 * np.bincount(y_train))
class_weights_dict = {0: class_weights[0], 1: class_weights[1]}

xgb_model = xgb.XGBClassifier(scale_pos_weight=class_weights_dict[1] / class_weights_dict[0], random_state
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

overall_accuracy_xgb = accuracy_score(y_test, y_pred_xgb) * 100
print("Overall Accuracy (XGBoost):  {:.2f}%".format(overall_accuracy_xgb))
print("XGBoost Model Report:")
print(classification_report(y_test, y_pred_xgb))

cm_xgb = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Useful', 'Useful'], yticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - XGBoost')
plt.show()
```

P 29.XGBoost code based on BERT feature vector in usefulness score model

LSTM

LSTM on TF-IDF feature vectors

In this study, I use an LSTM based on TF-IDF feature vectors for feature learning and prediction. First, I build an LSTM model to process TF-IDF feature vectors. In the model architecture, I use an embedding layer to map sparse TF-IDF features to a dense

embedding space. Next, I add two bidirectional LSTM layers, each containing 64 units, to capture contextual information in the text data. To prevent overfitting, I applied 50% dropout between the LSTM layers. Finally, the model was classified by a fully connected dense layer, which outputs the probability distribution of the categories using a softmax activation function. For the model compilation stage, I selected the adam optimizer and the categorical_crossentropy loss function, which is suitable for multi-class classification problems. During model training, I used 5 epochs and 64 batches, and applied class weights to deal with class imbalance. After the training was complete, I used the test data to make predictions and evaluated the model performance using classification_report and the confusion matrix. During training, I also plotted the training and validation loss and accuracy curves to observe the convergence of the model.

For further optimization, I increased the number of LSTM units and introduced L2 regularization to improve the generalization ability of the model. In addition, I set up an early stopping callback to prevent overfitting and restore the best weights. I optimized the model performance and generated the prediction results by increasing the number of training epochs to 10 and applying the same class

weights and batch sizes.

This sequence of steps ensured that the LSTM model could effectively learn from the TF-IDF features and perform accurate usefulness classification.

```
#  LSTM code Summary
# 1.LSTM model
# Build the LSTM model
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_shape=(100,)))
model.add(Bidirectional(LSTM(units=64, return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units=64)))
model.add(Dropout(0.5))
model.add(Dense(units=num_classes, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test), class_wei


# 2.LSTM Model, Increased units and L2 regularization
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_shape=(100,)))
model.add(Bidirectional(LSTM(units=128, return_sequences=True, kernel_regularizer='l2')))  # Increased uni
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units=128, kernel_regularizer='l2')))  # Increased units and L2 regularizatio
model.add(Dropout(0.5))
model.add(Dense(units=num_classes, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# 3. Set Up Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test),
                    class_weight={0: 1, 1: 5}, callbacks=[early_stopping])

# 5. Make Predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Define target names for classification report
target_names = [str(label) for label in label_encoder.classes_]

# 6. Print Model Evaluation Report
print("Overall Accuracy:  {:.2f}%".format(accuracy_score(y_test_classes, y_pred_classes) * 100))
print("LSTM model Report:")
print(classification_report(y_test_classes, y_pred_classes, target_names=target_names))

# 7. Plot Training Accuracy and Loss
plt.figure(figsize=(12, 6))
```

P 30.LSTM code based on TF-IDF feature vector in usefulness score model

LSTM on BERT feature vectors

In this study, I also applied BERT feature vector-based LSTM for feature learning and prediction. First, I designed a sequence model with bidirectional LSTM layers and Dropout layers to process the

text features extracted from BERT. The model was compiled using the Adam optimizer and binary cross-entropy loss function, and the training and validation losses and accuracy were recorded throughout the training process. To address the data imbalance problem, I used SMOTE to oversample the training data and calculated the class weights to balance the training process. Through grid search, I optimized the number of LSTM units and dropout rate and selected the hyperparameter configuration that performed best on the validation set. Finally, the evaluation results on the test set showed that my LSTM model performed well in the classification task, with a significant improvement in accuracy. The confusion matrix and classification report further verified the effectiveness of the model.

```
# LSTM code Summary
# 4.LSTM Model
lstm_model = Sequential()
lstm_model.add(Embedding(input_dim=5000, output_dim=128, input_shape=(bert_embeddings.shape[1],)))
lstm_model.add(Bidirectional(LSTM(units=32, return_sequences=True)))
lstm_model.add(Dropout(0.5))
lstm_model.add(Bidirectional(LSTM(units=32)))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(units=1, activation='sigmoid'))

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the LSTM model
lstm_history = lstm_model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))


# 5.LSTM+Smote, Class weights

# Use SMOTE to handle imbalanced data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Calculate class weights
class_weights = compute_class_weight('balanced', classes=np.unique(y_train_res), y=y_train_res)
class_weights_dict = dict(enumerate(class_weights))

# Expand dimensions of BERT embeddings to match LSTM input requirements
X_train_res = np.expand_dims(X_train_res, axis=1)  # Shape: (batch_size, 1, embedding_dim)
X_test = np.expand_dims(X_test, axis=1)            # Shape: (batch_size, 1, embedding_dim)


# Create LSTM model
lstm_model = Sequential()
lstm_model.add(Bidirectional(LSTM(units=32, return_sequences=True, input_shape=(X_train_res.shape[1], X_tr
lstm_model.add(Dropout(0.5))
lstm_model.add(Bidirectional(LSTM(units=32)))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
lstm_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

P 31.LSTM code based on BERT feature vector in the usefulness score model

4. Results and Evaluation

4.1 Evaluation Metrics

This study uses various evaluation metrics to comprehensively evaluate the performance of the sentiment analysis model. These metrics include (TP is the number of correct predictions in the positive class, TN is the number of correct predictions in the negative class, FP is the number of incorrect predictions in the negative class, and FN is the number of incorrect predictions in the positive class):

Accuracy

Accuracy measures the proportion of all samples that are correctly classified by the model. The calculation formula is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

Precision measures the accuracy of the model's predictions for the positive class. The higher the precision, the more accurate the model is in predicting the positive class. Its calculation formula is:

$$Precision = \frac{TP}{TP + FP}$$

Recall

Recall measures the model's ability to identify all positive classes.

The higher the recall, the higher the model's coverage in identifying positive classes. Its calculation formula is:

$$Recall = \frac{TP}{TP + FN}$$

F1 Score

The F1 score is the harmonic mean of precision and recall, which is used to comprehensively consider the accuracy and comprehensiveness of the model. Its calculation formula is:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Macro F1 Score：

Macro F1 score calculates the F1 score of each category and takes the average. This indicator focuses on the performance of all categories and is suitable for situations where the categories are unevenly distributed.

Weighted F1 Score：

Weighted F1 score calculates the F1 score for each category and takes a weighted average based on the number of samples in each category. This indicator better reflects the actual classification performance and is suitable for situations where the number of category samples varies greatly.

```
Overall Accuracy:  76.48%
Logistic Regression Model Report:
              precision    recall  f1-score   support

           0       0.78      0.92      0.84     15373
           1       0.70      0.43      0.54      7025

    accuracy                           0.76     22398
   macro avg       0.74      0.67      0.69     22398
weighted avg       0.76      0.76      0.75     22398
```

P 32.Example of an evaluation indicator

Confusion Matrix

The confusion matrix is a table used to evaluate the performance of a classification model. It shows how each predicted class relates to the actual class, including:

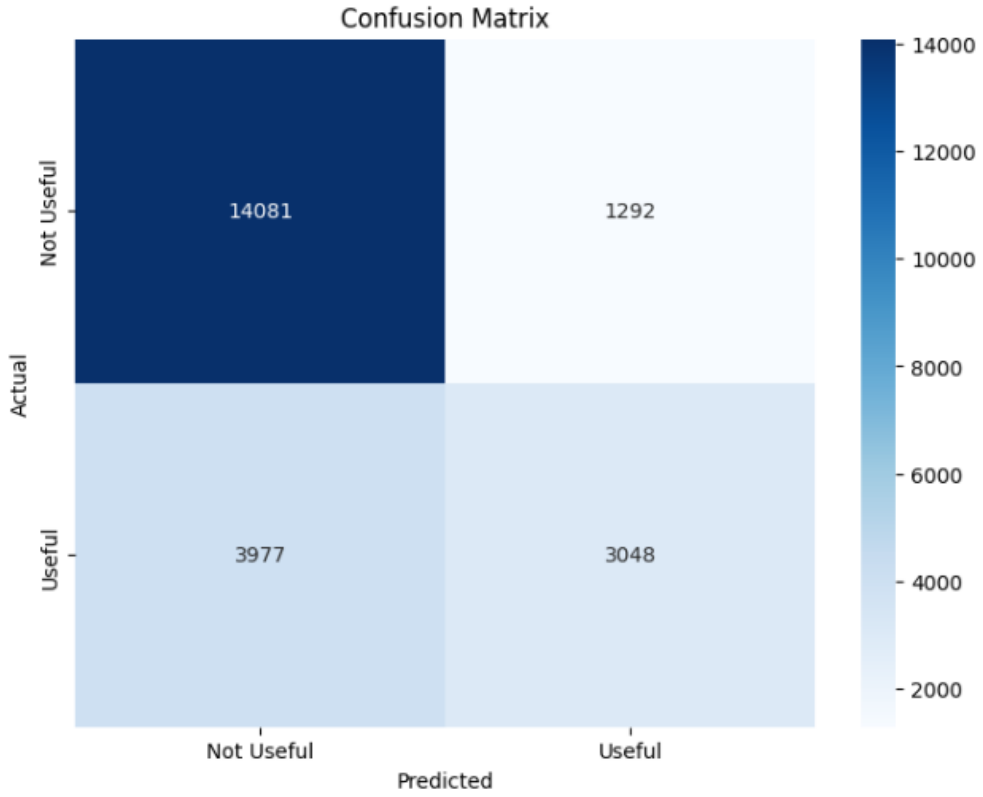
- TP(True Positive):The number of correctly predicted positive classes
- FP(False Positive):The number of incorrectly predicted positive classes.
- TN(True Negative): The number of correctly predicted negative classes.
- FN(False Negative): The number of incorrect predictions for the negative class.

The confusion matrix provides a visual representation of the categories in which the model performs well and those in which it performs poorly. For example, for the logistic regression and random forest models, I can use the code to generate a confusion matrix to

visually understand the classification effect of the model.

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Useful', 'Useful'], yticklabels=['No
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Generate usefulness score
df['usefulness_score'] = model.predict_proba(vectorizer.transform(df['cleaned_content']))[:, 1]
```

P 33.Confusion matrix code

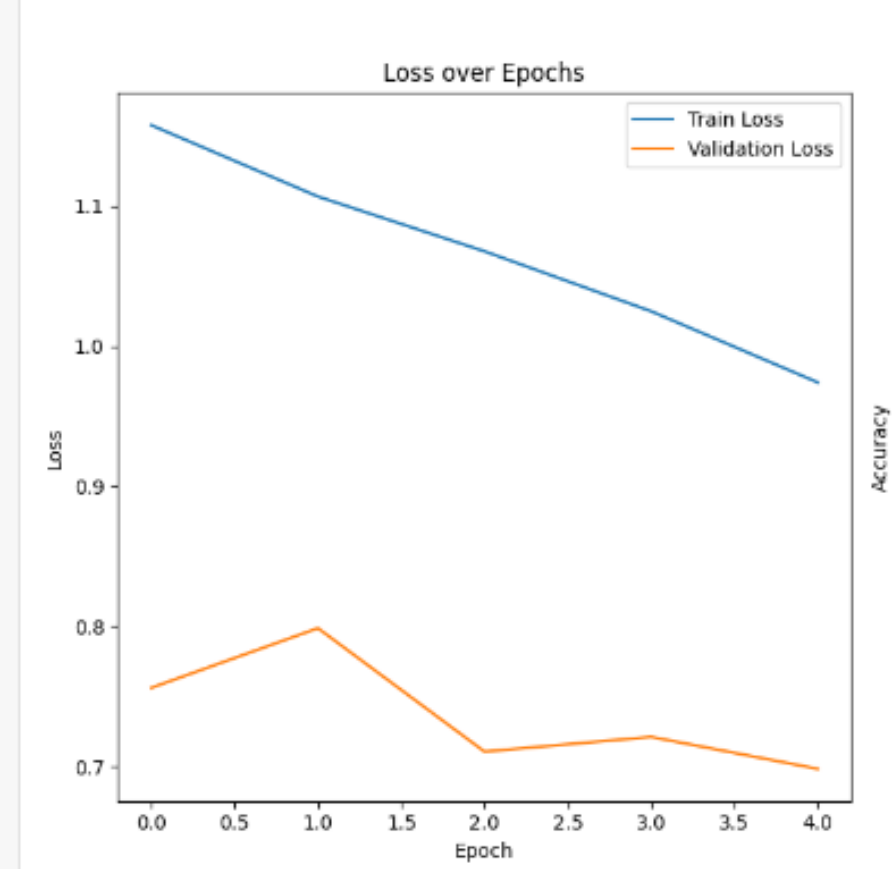


P 34.Confusion Matrix Visualization

Loss Curve

The loss curve shows the change in the loss value of the model during training and validation at each epoch. The training loss reflects the degree of fit of the model to the training data, while the validation loss evaluates the model's performance on unseen data. From this curve, we can visually observe the convergence of the

model. If the training loss continues to decrease while the validation loss increases, it may indicate overfitting of the model.

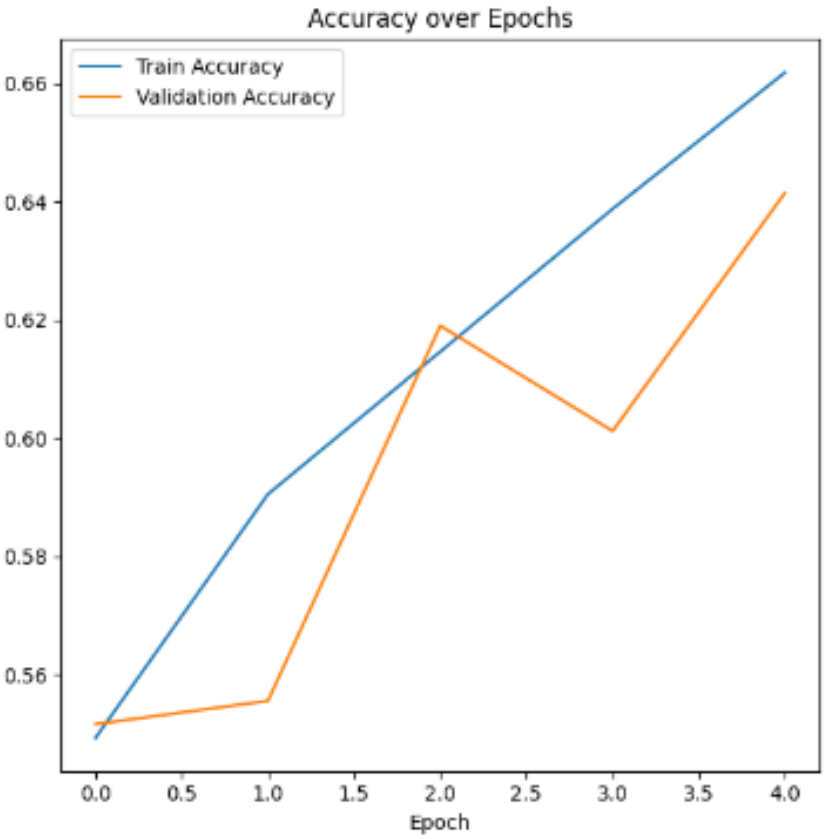


P 35. Loss Curve Visualization

Accuracy Curve

The accuracy curve shows the change in the model's accuracy on the training data and the validation data with each epoch. The training accuracy reflects the model's ability to predict the training data, while the validation accuracy is used to evaluate the model's performance on unseen data. By plotting the training and validation accuracy curves, you can observe the model's learning progress. A steady increase in accuracy generally indicates that the model's performance is continuously improving, while fluctuations in

accuracy may indicate that the model parameters or training strategy need to be adjusted.



P 36.Accuracy Curve Visualization

4.2 Sentiment Analysis Model Result

The performance of different models in the sentiment analysis task is demonstrated. The following are the specific results of the models under the TF-IDF and BERT feature extraction methods:

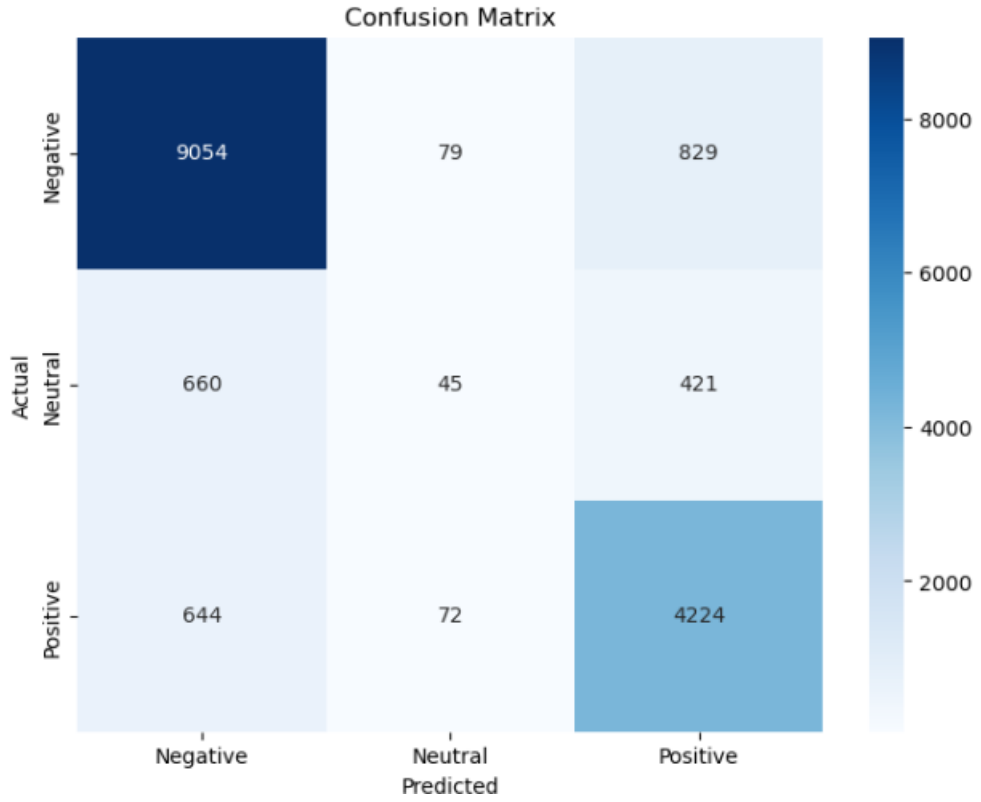
Performance of each model based on TF-IDF

Selected Model	Accuracy
Logistic Regression Model	83.12%
Logistic Regression Model +Class weight(neutral:3)	79.50%

Logistic Regression Model +Class weight(neutral:2)	81.73%
Logistic Regression Model + Hyperparameter Tuning (Grid Search)	81.57%
Logistic Regression Model + Feature extraction optimization (ngram_range=(1,2))	77.20%
Logistic Regression Model + Feature extraction optimization (ngram_range=(1,3))	76.92%
Ensemble Learning (Logistic Regression,Random Forest,Gradient Boosting)	81.30%
Ensemble Learning (Random Forest,Gradient Boosting,XGBoost)	80.96%
LSTM Model	79.83%
LSTM Model + Dropout	82.52%
LSTM Model + Dropout and BatchNormalization	81.85%
Bidirectional LSTM Model + Class weight	74.15%

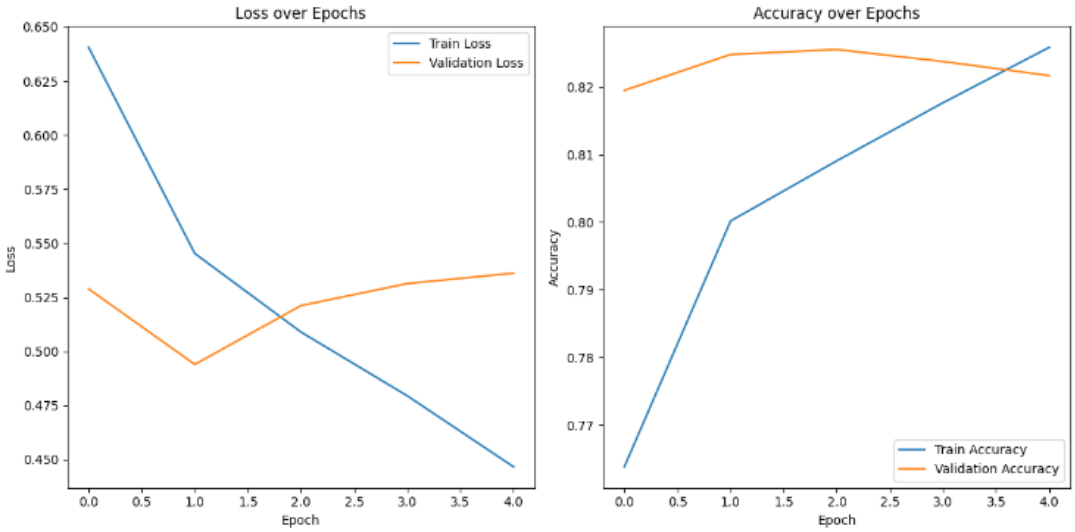
Table 1.Performance of each model based on TF-IDF

```
Overall Accuracy: 83.12%

Logistic Regression Model Report:
              precision    recall  f1-score   support

    Negative       0.87      0.91      0.89      9962
     Neutral       0.23      0.04      0.07      1126
    Positive       0.77      0.86      0.81      4940

    accuracy                           0.83     16028
   macro avg       0.63      0.60      0.59     16028
weighted avg       0.80      0.83      0.81     16028
```



P 37.Visualisation of the results of the logistic regression model based on TF-IDF in the sentiment analysis model

```
Overall Accuracy:  82.52%
LSTM Model + Dropout Report:
              precision    recall  f1-score   support

    Negative       0.85      0.92      0.88      9962
     Neutral       0.30      0.02      0.05      1126
    Positive       0.78      0.81      0.80      4940

    accuracy                           0.83     16028
   macro avg       0.64      0.59      0.58     16028
weighted avg       0.79      0.83      0.80     16028
```



P 38.Visualisation of the results of the TF-IDF-based LSTM model in the sentiment analysis model

Performance of each model based on Bert

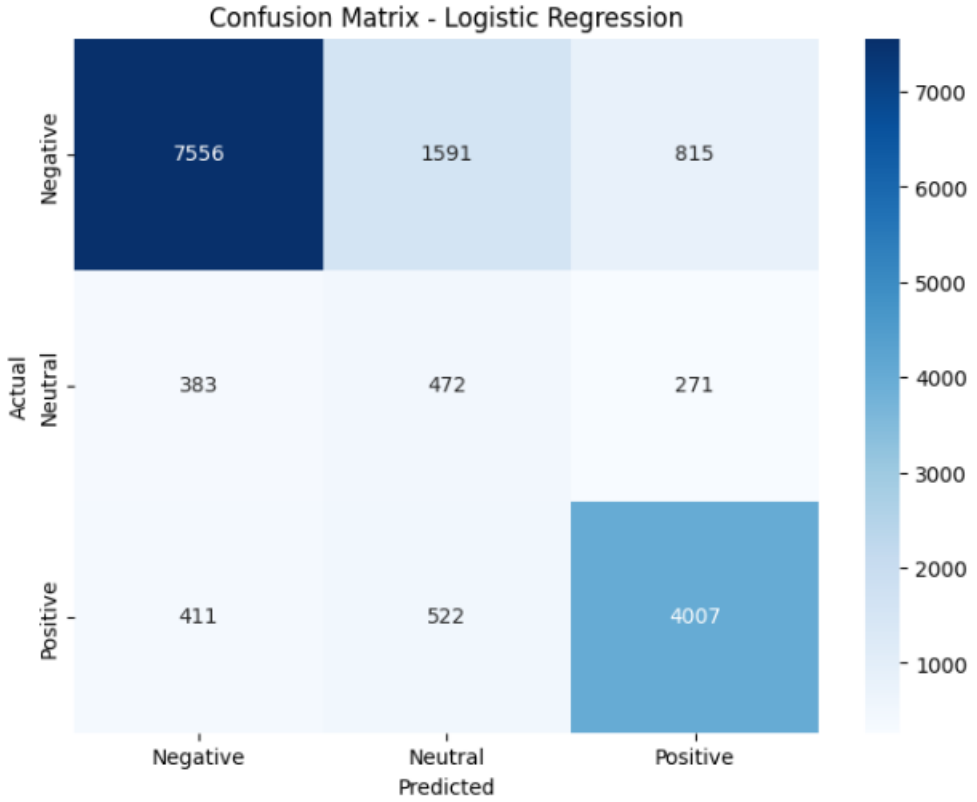
Selected Model	Accuracy
Logistic Regression Model	75.09%
Random Forest Model	79.60%
XGBoost Model	81.11%
LSTM Model	83.06%

Table 2.Performance of each model based on Bert

Overall Accuracy: 75.09%

Logistic Regression Model Report:

	precision	recall	f1-score	support
Negative	0.90	0.76	0.83	9962
Neutral	0.18	0.42	0.25	1126
Positive	0.79	0.81	0.80	4940
accuracy			0.75	16028
macro avg	0.62	0.66	0.63	16028
weighted avg	0.82	0.75	0.78	16028



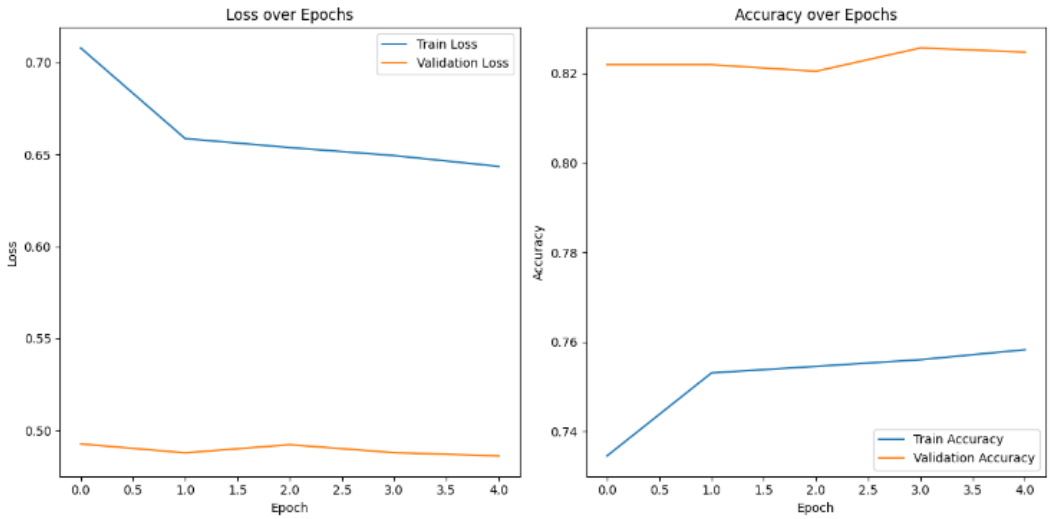
P 39.Visualisation of the results of the BERT-based logistic regression model in the sentiment analysis model

Overall Accuracy: 82.63%

LSTM Model + Dropout and BatchNormalization Report:

	precision	recall	f1-score	support
Negative	0.85	0.92	0.88	9962
Neutral	0.00	0.00	0.00	1126
Positive	0.77	0.84	0.80	4940
accuracy			0.83	16028
macro avg	0.54	0.58	0.56	16028
weighted avg	0.77	0.83	0.80	16028

P 40.Visualisation of the results of the BERT-based LSTM model in the sentiment analysis model



P 41.Loss and Accuracy Curve of the BERT-based LSTM model in the Sentiment Analysis model

Result

Among all models, the LSTM model with BERT feature vectors performed best, with an overall accuracy of 83.06%. For the classification of negative sentiment, the LSTM model with Dropout and BatchNormalization achieved the highest F1 score of 0.88. The LSTM model with BERT feature vectors and the LSTM model + Dropout performed equally well in the classification of positive sentiment, both achieving an F1 score of 0.80. However, no model performed well in the neutral sentiment classification, and the LSTM model with BERT feature vectors and the XGBoost model performed the worst in this category. Considering the overall accuracy and performance in each category, the LSTM model with BERT feature vectors performed well overall and in the classification of negative sentiment. Although the model's performance in the

neutral category was not satisfactory, further optimization work may include a more in-depth analysis or data expansion of neutral category samples or attempts to improve its performance with more complex model structures, considering that the neutral category performed poorly among all models.

4.3 Usefulness Score Model Result

This article shows how different models perform in the usefulness scoring task. The following are the specific results of the models under the TF-IDF and BERT feature extraction methods:

Performance of each model based on TF-IDF

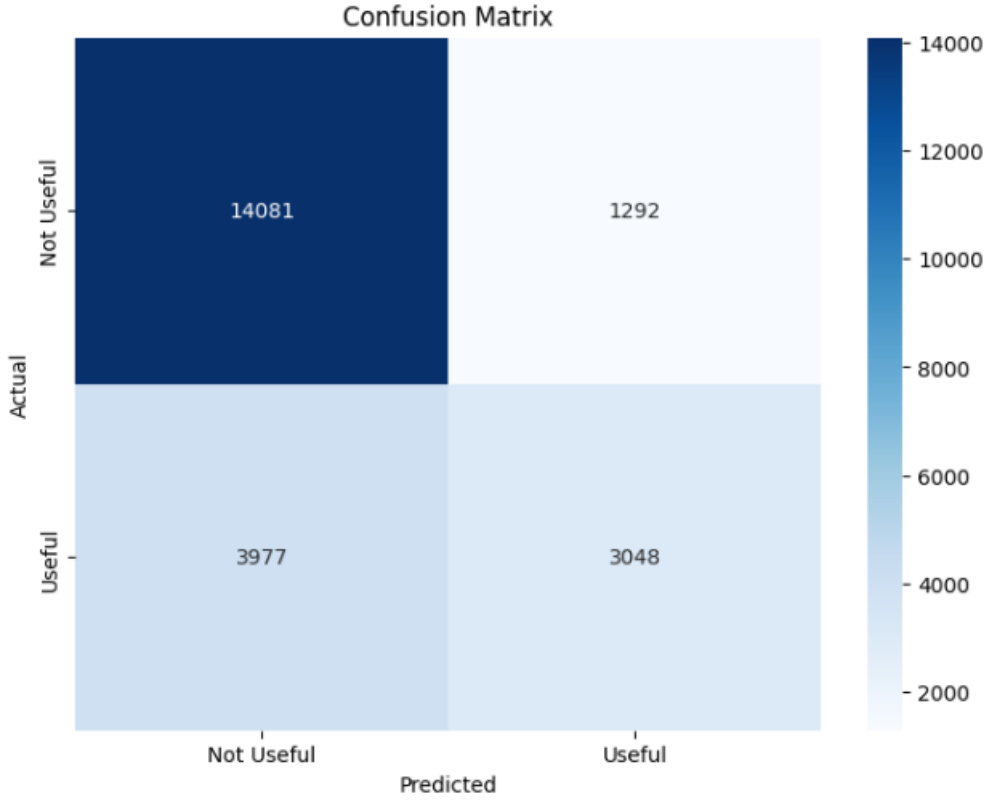
Selected Model	Accuracy
Logistic Regression Model	76.48%
Logistic Regression Model +class_weight=balanced	72.47%
Logistic Regression Model + Increasing Iterations	68.64%
Random Forest Model	75.21%
Random Forest Model + Class Weight	75.39%
Random Forest Model + GridSearchCV	73.69%
XGBoost Model	75.39%
XGBoost Model + GridSearchCV	73.69%
LSTM Model	64.15%
LSTM Model + Increase units and L2 regularization	66.02%

Table 3.Performance of each model based on TF-IDF

Overall Accuracy: 76.48%

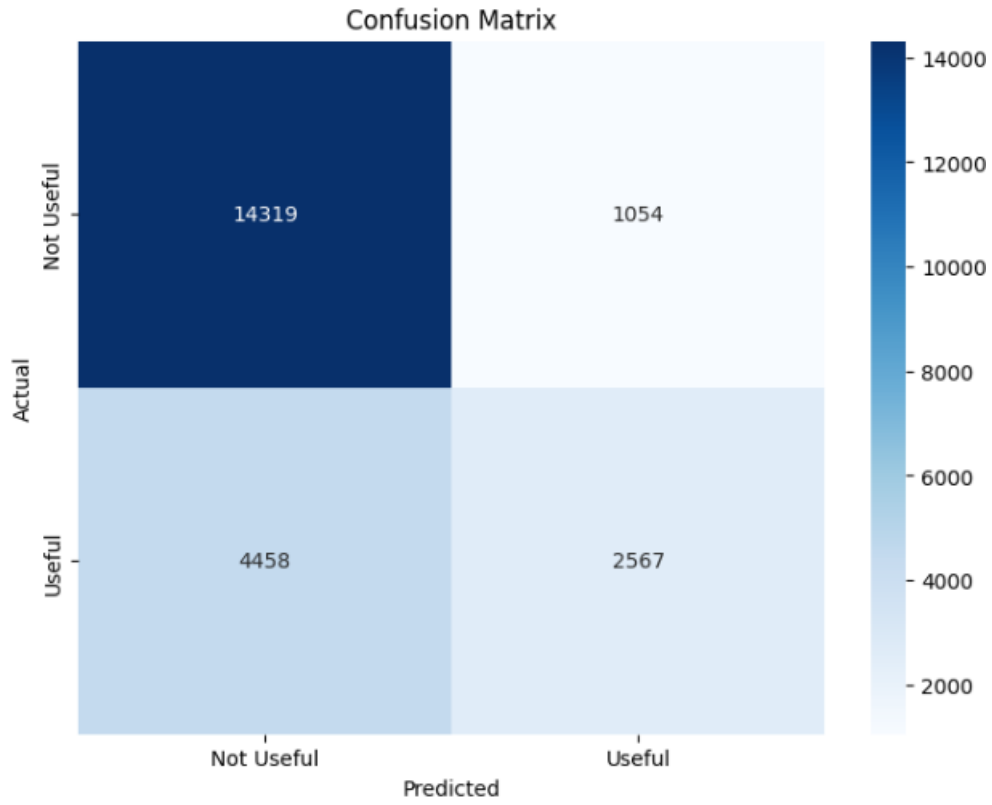
Logistic Regression Model Report:

	precision	recall	f1-score	support
0	0.78	0.92	0.84	15373
1	0.70	0.43	0.54	7025
accuracy			0.76	22398
macro avg	0.74	0.67	0.69	22398
weighted avg	0.76	0.76	0.75	22398



P 42.Visualisation of the TF-IDF based logistic regression model in the usefulness score model

```
Overall Accuracy:  75.39%
Random Forest Model + Class Weight Report:
              precision    recall  f1-score   support

           0       0.76      0.93      0.84     15373
           1       0.71      0.37      0.48      7025

    accuracy                           0.75     22398
   macro avg       0.74      0.65      0.66     22398
weighted avg       0.75      0.75      0.73     22398
```



P 43.Visualisation of the TF-IDF random forest model in the usefulness score model

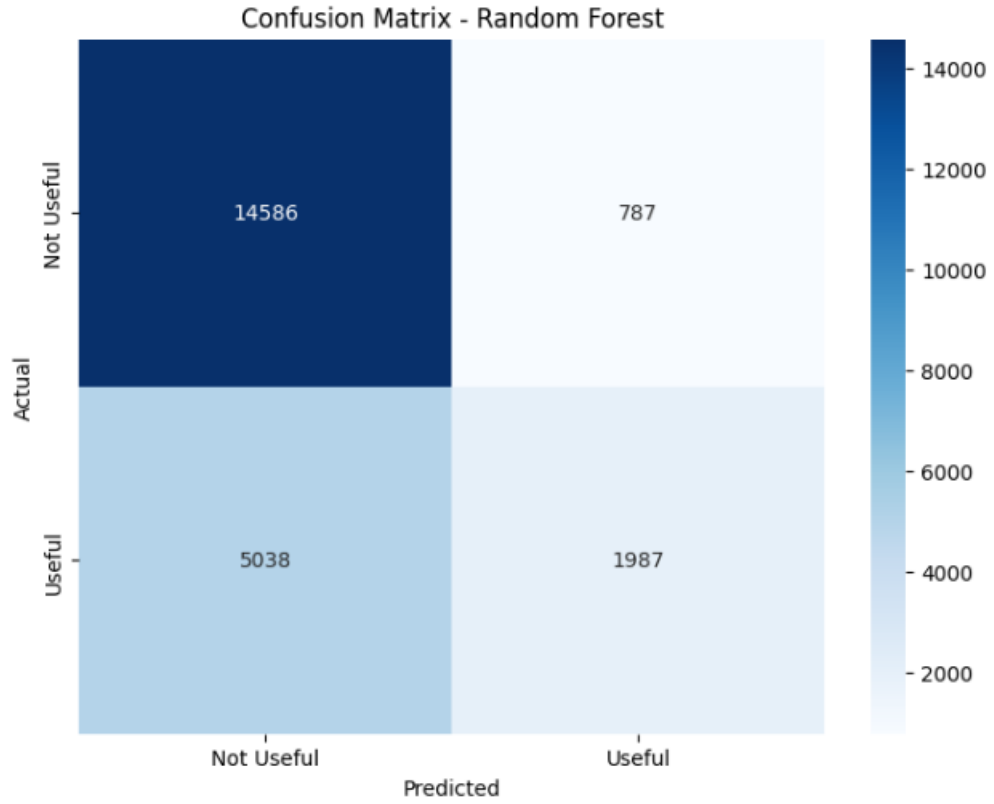
Performance of each model based on Bert

Selected	Accuracy
Logistic Regression Model	71.79%
Random Forest Model	73.99%
XGBoost Model	71.05%
LSTM Model	68.64%
LSTM Model + Smote, Class weights	74.70%

LSTM Model + Smote, Class weights2	75.25%

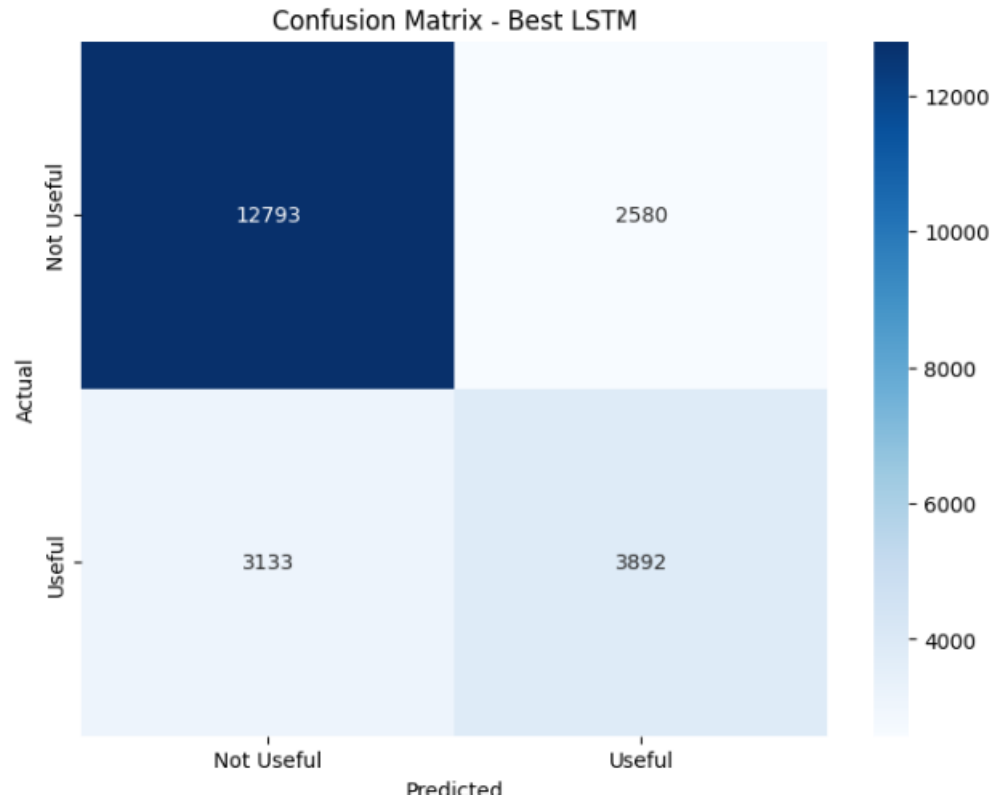
Table 4.Performance of each model based on Bert

```
Overall Accuracy (Random Forest):  73.99%
Random Forest Model Report:
              precision    recall  f1-score   support

           0       0.74      0.95      0.83     15373
           1       0.72      0.28      0.41      7025

    accuracy                           0.74     22398
   macro avg       0.73      0.62      0.62     22398
weighted avg       0.73      0.74      0.70     22398
```



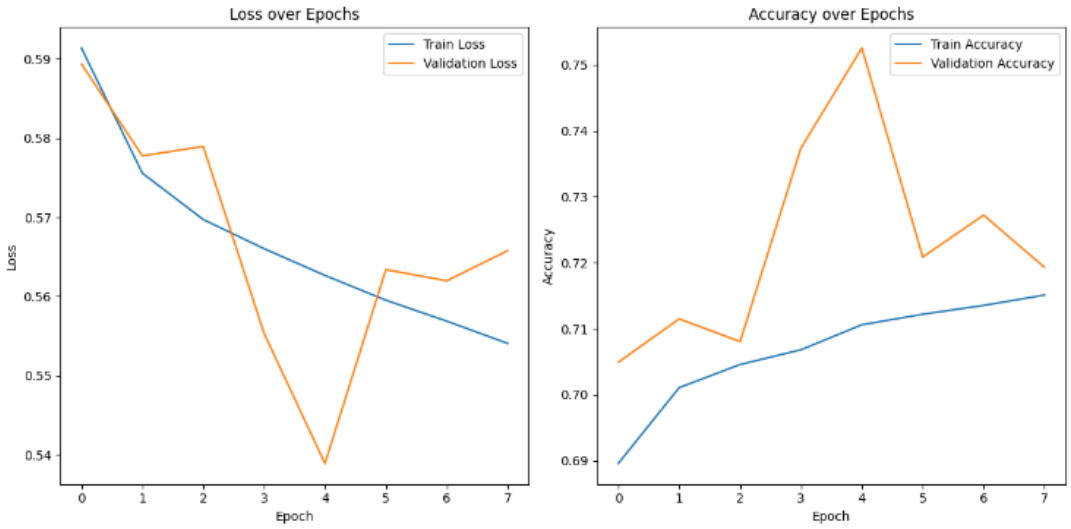
P 44.Visualisation of the BERT-based random forest model in the usefulness score model

```
Overall Accuracy (Best LSTM):  74.49%
Best LSTM Model Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.82     15373
           1       0.60      0.55      0.58      7025

    accuracy                           0.74     22398
   macro avg       0.70      0.69      0.70     22398
weighted avg       0.74      0.74      0.74     22398
```



P 45.Visualisation of the BERT-based LSTM model in the usefulness score model

```
Accuracy: 0.7419
Best Parameters: {'dropout_rate': 0.5, 'units': 16}
Best Accuracy: 0.7525
```



P 46.Loss and Accuracy Curve of the BERT-based LSTM model in the usefulness score model

Result

Overall, the models based on BERT feature representations performed better, especially the combination of BERT and LSTM (accuracy 74.70%) and BERT and random forest (accuracy 73.99%). These models recognize both categories more equally in the classification task, especially in category 1. In contrast, the TF-IDF-based models perform slightly worse, especially in the classification task for category 1. Although the performance of random forest and XGBoost is acceptable, the overall effect is not optimal.

5. Critical Discussion

5.1 Comparison of Model Performance

In the sentiment analysis and usefulness scoring tasks, although the BERT+LSTM model excels at processing long texts and complex semantics, with accuracy and F1 scores higher than the traditional TF-IDF method, its superiority also comes with some limitations. The BERT+LSTM model uses the pre-trained language expression ability of BERT and the sequence modelling ability of LSTM to effectively capture the deep information and long-term dependencies in the text, greatly improving the model's sentiment understanding ability and practicality. However, the complexity of this model also brings an increase in computational overhead, and the training and inference times are significantly longer than those of the TF-IDF-based method. In addition, although the BERT+LSTM model performed best in terms of overall accuracy and negative sentiment classification, its performance in neutral sentiment classification was still unsatisfactory, with no significant improvement compared to the XGBoost model and other models. This indicates that when dealing with certain categories, the BERT+LSTM model may need to be further adjusted or combined with other techniques to improve the classification effect. Therefore,

although the BERT+LSTM model has strong text processing capabilities, in practical applications, it is still necessary to balance its complexity and performance, and optimize the model according to the requirements of specific tasks.

5.2 Comparison of Feature extraction methods

The TF-IDF method excels in text data feature extraction, mainly because it can effectively extract word frequency information and the relative importance of words. However, its limitation is that it ignores contextual relationships and semantic information. In contrast, the BERT method captures richer contextual information and semantic relationships through a pre-trained deep language model, so it has a clear advantage in feature representation.

Although BERT models have higher computational overheads during training and inference, requiring more time and resources, the refined feature representations they provide can significantly improve the accuracy and reliability of sentiment analysis and usefulness scoring tasks. However, BERT's high computational complexity and resource requirements may pose challenges for practical applications, and it is necessary to weigh its efficiency and applicability. In terms of the choice of feature extraction method, although BERT has a clear advantage in terms of accuracy, for

scenarios with limited resources, TF-IDF is still an effective option. Therefore, in different application scenarios, it is necessary to comprehensively consider model performance, computing resources and actual needs to select the most appropriate feature extraction method.

5.3 Analysis of the effectiveness of LSTM architecture on the problem of training/validation loss fluctuation

In this paper, I found that in the process of building an affective analysis and usefulness scoring model, the training/validation loss of the neural network has a problem with spikes. This indicates that at some stage the model is overfitting or underfitting the data.

LSTM can effectively alleviate this problem through its unique mechanism. First, the gating mechanism of LSTM can selectively remember or forget information, thereby reducing the problems of gradient disappearance and gradient explosion, making the training process smoother and avoiding sudden spikes in the loss function. Second, LSTM is good at handling dependencies in long sequence data. Especially in tasks such as sentiment analysis, the emotional meaning of a word often depends on the previous and next context. LSTM can effectively capture this contextual dependency, thereby

reducing the occurrence of loss spikes. In addition, the memory units of LSTM can retain important information over multiple time steps, which is particularly important for tasks such as sentiment analysis and usefulness scoring that deal with long texts or complex semantics. This allows the model to better remember long-distance dependencies related to sentiment or usefulness, thereby improving the stability and performance of the model.

Data often contains noise or class imbalance (e.g., a small number of neutral class samples in sentiment analysis), which can also cause the training/validation loss to peak. I have combined a series of techniques to optimize this process. The application of dropout and batch normalization can effectively reduce overfitting, making the model training process more stable for data from different batches and reducing fluctuations in the validation loss. In addition, to address the problem of data imbalance, by assigning class weights and using SMOTE, the LSTM model can learn minority classes more effectively, further reducing the instability of the validation loss. The multi-layer LSTM architecture has been designed to enhance the performance of the model. It extracts features layer by layer through multi-layer information abstraction, which can better adapt to complex input data. In particular, the use

of bidirectional LSTM helps to capture both the forward and backward information of the text at the same time, thereby improving semantic understanding. The use of a learning rate scheduler (ReduceLROnPlateau) and an early stopping mechanism (EarlyStopping) helps prevent loss function oscillations and overfitting caused by excessively long training times, further smoothing the training process and reducing the peak of the validation loss. The combination of these techniques makes the LSTM more robust when dealing with complex data and imbalances, thereby enhancing the stability of the model.

5.4 Advantages and Disadvantages of Ensemble Pretrained Embeddings in LSTM

In building the sentiment analysis model and usefulness scoring model, we used the pre-trained embedding generated by BERT as the input layer of the LSTM model and explored the advantages and disadvantages of integrating the pre-trained embedding into the LSTM model.

In contrast to randomly initialized embedding layers, which only provide fixed and static word vectors and cannot dynamically adjust or capture contextual information, BERT pre-trained embedding layers exhibit excellent contextual understanding capabilities.

Because BERT uses a bidirectional transformer architecture, it can consider both the left and right sides of the context when generating word embeddings. This contextual awareness is particularly important for the sentiment analysis task of LSTM, as sentiment often depends on the subtle relationships between words. For example, BERT can distinguish between the different meanings of ‘very’ in ‘very good’ and ‘very bad’, thus capturing sentiment more accurately.

Randomly initialized embedding layers typically require a large amount of training data to optimize their semantic expressiveness. TF-IDF embeddings are mainly based on word frequency and document frequency, lacking a deep understanding of word meaning and context. In contrast, integrating BERT pre-trained embeddings can capture richer semantic information. BERT is pre-trained on large-scale corpora, so it can provide deeper and more accurate semantic representations, enabling LSTM to more effectively use these embeddings for sentiment classification, thereby improving model performance.

In terms of training efficiency, the randomly initialized embedding layer requires more training time and data to optimize, while the TF-

IDF embedding has a lower computational overhead but usually needs to be combined with other features or models to achieve better results. In contrast, BERT has been pre-trained on a large amount of data, so directly using its embeddings can greatly reduce the training time and data requirements. The LSTM model only needs to fine-tune the pre-trained BERT embeddings, without having to train the embedding layer from scratch.

Although there are many advantages to integrating BERT pre-trained embeddings, there are also some disadvantages that should not be ignored. First, BERT has high computational and memory overheads, especially when processing long texts or large-scale data. It usually requires high computational resources and storage space, which may limit its application in resource-limited environments. In contrast, randomly initialized embedding layers and TF-IDF embeddings have lower computational costs and are more suitable for scenarios with limited computing resources. However, they are lacking in terms of feature richness and contextual understanding. In addition, the BERT pre-trained corpus may be different from the specific task domain, so in some cases additional domain fine-tuning is required to better meet the needs of a specific task. The randomly initialized embedding layer can be

trained from scratch for a specific task, although this requires a large amount of domain-specific data to achieve the best results.

In summary, integrating BERT pre-trained embedding layers into LSTM models can significantly improve the model's contextual understanding and semantic expression richness, while reducing training time and data requirements. However, this approach also faces high computational and memory overheads, as well as potential domain adaptation issues. In contrast, although the computational cost of randomly initialized embedding layers and TF-IDF embeddings is lower, their semantic expression and contextual understanding capabilities are relatively limited. Therefore, it is important to choose the appropriate embedding method based on the specific application scenario and resource constraints, to improve performance while considering the feasibility of computational resources.

5.5 Comparative analysis of BERT misclassification and TF-IDF correct classification in sentiment analysis

In the sentiment analysis task and the usefulness rating task, the BERT model and the TF-IDF model showed significant differences in handling different types of comments. In particular, for the

comments that were misclassified by the BERT model but correctly classified by the TF-IDF model, we explored the commonalities and analyzed the reasons, thus revealing the advantages and disadvantages of the two models in task analysis.

First, in some reviews, the BERT model misclassifies negative sentiment as neutral or positive sentiment. For example, reviews such as ‘sexual content’ or ‘login problems’ are sometimes misjudged as neutral or positive by the BERT model, even though they carry a clear negative sentiment. The reason for this misclassification may be that the BERT model focuses more on the overall context when dealing with complex semantics and contextual relationships, while ignoring clear negative indicators. For example, ‘sexual content’ may be interpreted as neutral or a general description, and BERT may not fully capture the underlying negative sentiment due to its deep semantic understanding. In contrast, the TF-IDF model is more sensitive to direct negative words such as ‘problem’ or ‘issue’ by calculating the frequency and characteristics of words, and therefore can accurately identify these negative sentiments.

BERT models also often misclassify comments that refer to specific

technical issues or functional limitations. For example, for comments such as ‘login issues’ or ‘netflix app issues,’ BERT may fail to correctly identify the sentiment due to a complex understanding of the context. Such comments often have clear negative indicators such as ‘problem’ or ‘issue’, and the BERT model may try to understand the context more deeply but ignore these direct negative indicators. The IDF model relies on the frequency characteristics of words, and for these types of comments that clearly express negative emotions, it can more quickly and consistently capture sentiment tendencies and accurately classify them.

The BERT model is also limited in its performance when faced with short or ambiguous comments. For example, many comments are very short, consisting of just one word or phrase (e.g. ‘app’, ‘swahili’, ‘pradeepd’). These comments often lack context and provide little semantic information, making it difficult for BERT to accurately identify emotions. For example, the word ‘app’ may represent different meanings and emotional tendencies in different contexts. The TF-IDF model, on the other hand, does not rely on context and is based solely on the frequency and importance of word occurrences, so it can more accurately identify emotions when

processing these short texts.

In addition, the BERT model does not perform well when dealing with misspellings or non-standard terms. For example, comments containing spelling errors, phonetic transcriptions or non-standard expressions such as ‘bullshiy’, ‘ngi ta ng k nguyn ti khon’ or ‘balik niyo og trigun’ deviate from the corpus on which the BERT model was trained, making it difficult to accurately parse and understand their sentiment. The BERT model relies on a large amount of training data to learn the complex relationships between language structures and vocabulary. If the words in the comment do not match the training data, BERT may not be able to correctly identify the sentiment. In contrast, the TF-IDF model processes text by calculating the frequency of words in the text and is not related to these linguistic variants. It is therefore better able to handle comments containing spelling errors or romanizations.

The BERT model also shows its limitations when dealing with multilingual or mixed-language comments. For example, comments such as ‘bahut acchi movie’ (in Hindi) or ‘no me gusta la nueva actualización’ (in Spanish) are multilingual or contain mixed expressions of multiple languages. The TF-IDF model does not rely

on specific language rules and only considers the frequency of word occurrences and their distribution characteristics, so it can better adapt to linguistic diversity and make more accurate sentiment classifications.

The BERT model's performance is also challenged by comments that contain multiple emotions or mixed opinions. For example, ‘i hate ads so much... but anywho netflix is lovely’ (I really hate ads... but anyway, netflix is lovely) expresses both a strong negative emotion (‘i hate ads so much’) and a positive emotion (‘netflix is lovely’), which makes sentiment classification more complicated. The BERT model is good at processing complex sentence structures and semantic information, but because it is highly dependence on the context, it may be misled by certain emotional factors, resulting in classification errors. The TF-IDF model relies on word frequency for sentiment analysis, so it can more accurately capture sentiment trends when faced with comments containing obvious emotional words, especially when negative emotional words appear more frequently.

Finally, the BERT model also faces challenges when dealing with sarcastic, metaphorical or ambiguous expressions of sentiment. For

example, comments such as ‘idiot box app’ or ‘pro zionist app’ contain sarcastic or complex expressions of sentiment, and the BERT model is prone to misclassifications due to its ability to parse context and deep semantics. The BERT model is prone to misjudgments due to its susceptibility to context. For example, the sentiment of the word ‘idiot’ in the sentence ‘idiot box app’ may be perceived as neutral or negative depending on the context. In the absence of a clear context, BERT may not be able to make an accurate judgment. The IDF model directly captures prominent emotional words through word frequency features, so it performs better in these comments with implicit emotions.

In short, BERT and TF-IDF models each have their own advantages and disadvantages in analysis. The BERT model performs well when dealing with complex contexts and semantic structures but is prone to misjudgments when faced with short texts, irregular language, multilingual comments, mixed emotions or expressions with a satirical tone. The TF-IDF model, on the other hand, is more stable and accurate in these situations due to its straightforward representation of word frequency features. The combined use of these two models is expected to enhance the overall analysis results.

```
Reviews where BERT LSTM is wrong but TF-IDF LSTM is correct:

Review: i do not liked it at all
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 2 (Label: Positive)
TF-IDF LSTM Prediction: 2 (Label: Positive)

Review: i thought that all of the old and new movie is in the netflix
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 2 (Label: Positive)
TF-IDF LSTM Prediction: 2 (Label: Positive)

Review: pro zionist app
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 2 (Label: Positive)

Review: who am i think and awnser me
True Sentiment: 2 (Label: Positive)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: login problem login atemet
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: so gooodddd
True Sentiment: 2 (Label: Positive)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: please challenge kar do
True Sentiment: 2 (Label: Positive)
BERT LSTM Prediction: 2 (Label: Positive)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: jaana jeele apni jindagi
True Sentiment: 2 (Label: Positive)
```

P 47.Reviews where BERT LSTM is wrong, but TF-IDF LSTM is correct

6. Conclusion

This paper provides a comprehensive evaluation of various models for the sentiment analysis and usefulness rating tasks. The results show that a combination of BERT and LSTM models achieves the best results for both tasks. The contextual understanding of BERT and the sequence modelling ability of LSTM complement each other, making this deep learning approach significantly better than other models when dealing with complex text features.

In the sentiment analysis task, the BERT+LSTM model outperformed the traditional TF-IDF method in terms of accuracy and F1 score. This shows that deep learning methods are superior at capturing subtle sentiment information in reviews. LSTM is good at processing time series data, while BERT can understand complex contextual relationships in depth, thereby improving the model's understanding of long texts and prediction efficiency.

Similarly, in the usefulness rating task, the BERT+LSTM model showed a significant improvement in accuracy and F1 score compared to TF-IDF combined with logistic regression and random forest models. This shows that the deep learning model not only understands the content of the review more accurately, but also

predicts the actual usefulness of the review more effectively.

In the comparative analysis, we found that although BERT+LSTM excels at handling complex contexts, it sometimes classifies certain reviews less accurately than TF-IDF. For example, because BERT+LSTM relies on in-depth contextual understanding, it may misinterpret negative or ambiguous comments, while the simple approach of TF-IDF is more reliable in these cases. This highlights the need to combine different models to leverage their respective strengths and improve overall analysis results.

Although the BERT+LSTM model has a clear advantage in terms of performance, it also has a relatively high computational overhead and takes significantly longer to train and infer than the TF-IDF method. Future research should focus on exploring more advanced deep learning techniques, such as newer models based on the Transformer architecture, which have the potential to provide more refined feature representations. In addition, it is worth further investigating the combination of transfer learning with pre-trained models to improve model performance and adaptability. To improve model generalization, various data augmentation methods and optimization strategies should be employed while balancing

performance and computational resource requirements. By applying these advanced techniques, the effectiveness and efficiency of the model can be further improved.

7. Reflection

In this project, I faced many challenges and learning opportunities, including the application of TF-IDF, logistic regression, random forest, XGBoost, and more advanced BERT and LSTM models.

Previously, I had only been exposed to simple logistic regression and random forest on SAS software and lacked practical experience in operating complex models. To learn these techniques efficiently, I made a detailed learning plan, breaking down the learning tasks into small pieces and setting daily goals to maintain consistency and motivation. At the same time, I used various resources such as online courses, official documents and open-source projects (such as Coursera, TensorFlow, etc.) to apply theoretical knowledge to practical problems through practice and experimentation, so as to improve learning effectiveness.

First, although TF-IDF is the most basic feature extraction method, how it is combined with traditional machine learning is still a challenge for me.

Second, the pre-trained model and embedding mechanism of BERT are quite complex, and it is difficult for beginners to understand its

working principle and architecture. The bidirectional Transformer structure and self-attention mechanism of BERT involve profound theories that take a long time to digest. In addition, BERT has high computational requirements, and the resource consumption of training and inference is much higher than that of traditional models, which requires a deeper understanding of hardware configuration and resource management. BERT needs to process a large amount of text data and long sequences, which makes model adjustment and optimization more complicated.

The internal mechanism of LSTM is relatively complex, including the operation of the forget gate, input gate and output gate. For beginners, the operation of these gated mechanisms is difficult to understand and debug. The training process of the LSTM model is likely to lead to overfitting and requires careful adjustment. In addition, experience is still needed to effectively select and design the structure of the LSTM network (such as the number of layers and units). Due to time constraints, I have not fully mastered BERT and LSTM, and the actual operation and adjustment is more complicated than I thought.

If I were to do this project from scratch, I would spend more time

researching the technology at the beginning of the project, systematically learning about NLP, machine learning and deep learning, including gaining a deeper understanding of how BERT and LSTM work and their application examples. I would also consider introducing more model evaluation and adjustment methods to improve the accuracy and robustness of the final model. In addition, I would join forums such as Stack Overflow to learn from other people's experiences and seek help.

In short, learning and applying NLP techniques and deep learning models is a challenging process, but with effective time management and practical experience, these difficulties can be overcome. By summarizing the lessons learned, I can better plan future research and development work and continuously improve my technical capabilities and project management skills.

Reference

Tomaiuolo, N.G., 2012. UContent: The information professional's guide to user-generated content. Medford, New Jersey: Information Today.

Dang, N.C., Moreno-García, M.N. and De la Prieta, F., 2020. Sentiment Analysis Based on Deep Learning: A Comparative Study. Electronics, 9(3).

Shan, C., Wang, H., Chen, W., & Song, M. (2015). The Data Science Handbook. Data Science Bookshelf, New York.

Svolba, G., 2006. Data Preparation for Analytics Using SAS. Cary, NC: SAS Institute.

Salton, G. and Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24(5).

Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language

Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019).

Pampel, F.C., 2000. Logistic Regression: A Primer. Sage Publications.

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. R news, 2(3).

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).

Appendices

Appendix A: Dataset link and Code link

Dataset link:

https://www.kaggle.com/datasets/ashishkumarak/netflix-reviews-playstore-daily-updated

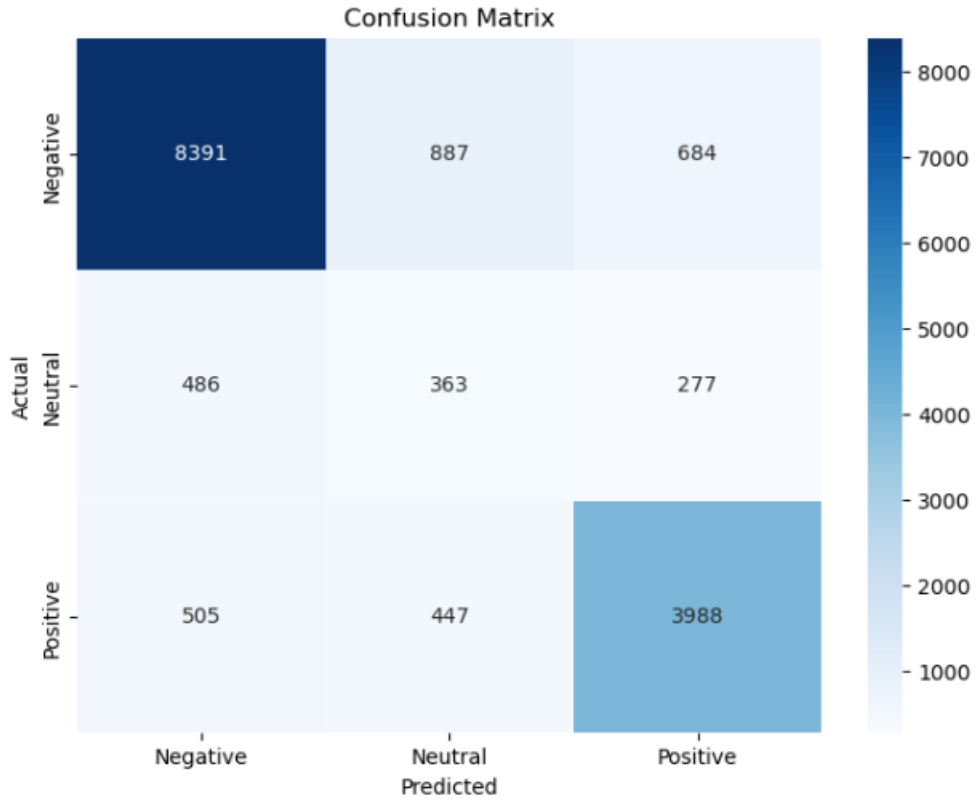
Code link:

https://github.com/Kerryru/Data-analysis

Appendix B: Model Output Result:

TF-IDF in sentiment analysis models

```
Overall Accuracy: 79.50%

Logistic Regression Model + Class Weight Report1:
              precision    recall  f1-score   support

    Negative       0.89      0.84      0.87      9962
     Neutral       0.21      0.32      0.26      1126
    Positive       0.81      0.81      0.81      4940

    accuracy                           0.79     16028
   macro avg       0.64      0.66      0.64     16028
weighted avg       0.82      0.79      0.81     16028
```

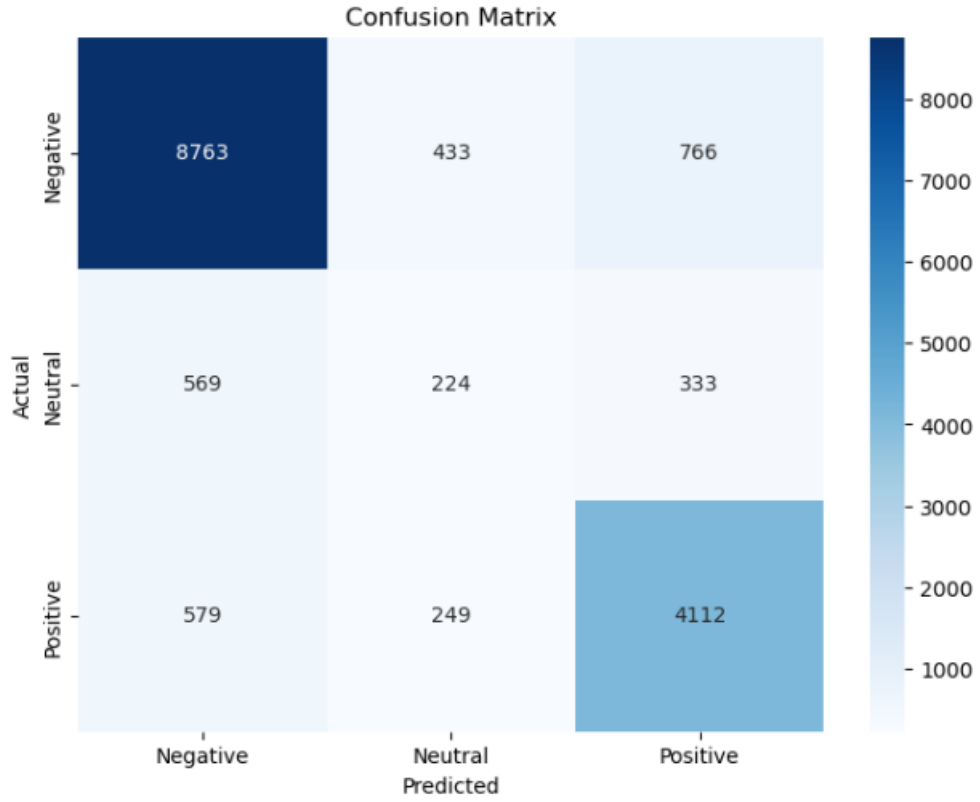


Logistic Regression Model + Class Weight

Overall Accuracy: 81.73%

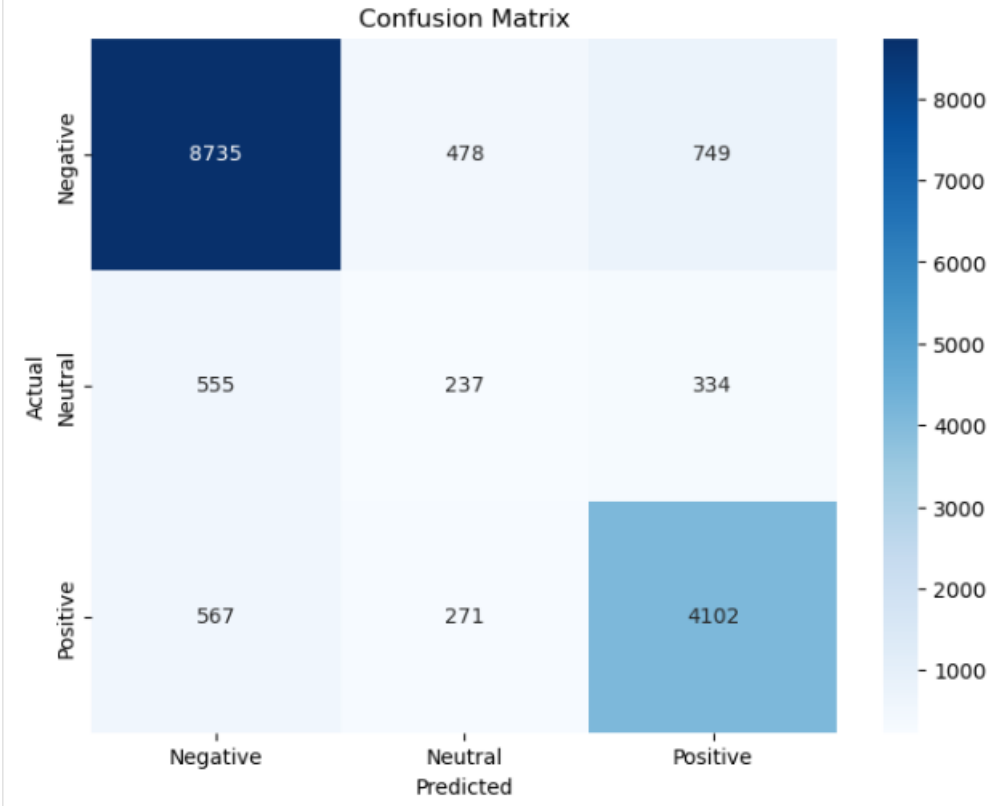
Logistic Regression Model + Class Weight Report2:

	precision	recall	f1-score	support
Negative	0.88	0.88	0.88	9962
Neutral	0.25	0.20	0.22	1126
Positive	0.79	0.83	0.81	4940
accuracy			0.82	16028
macro avg	0.64	0.64	0.64	16028
weighted avg	0.81	0.82	0.81	16028



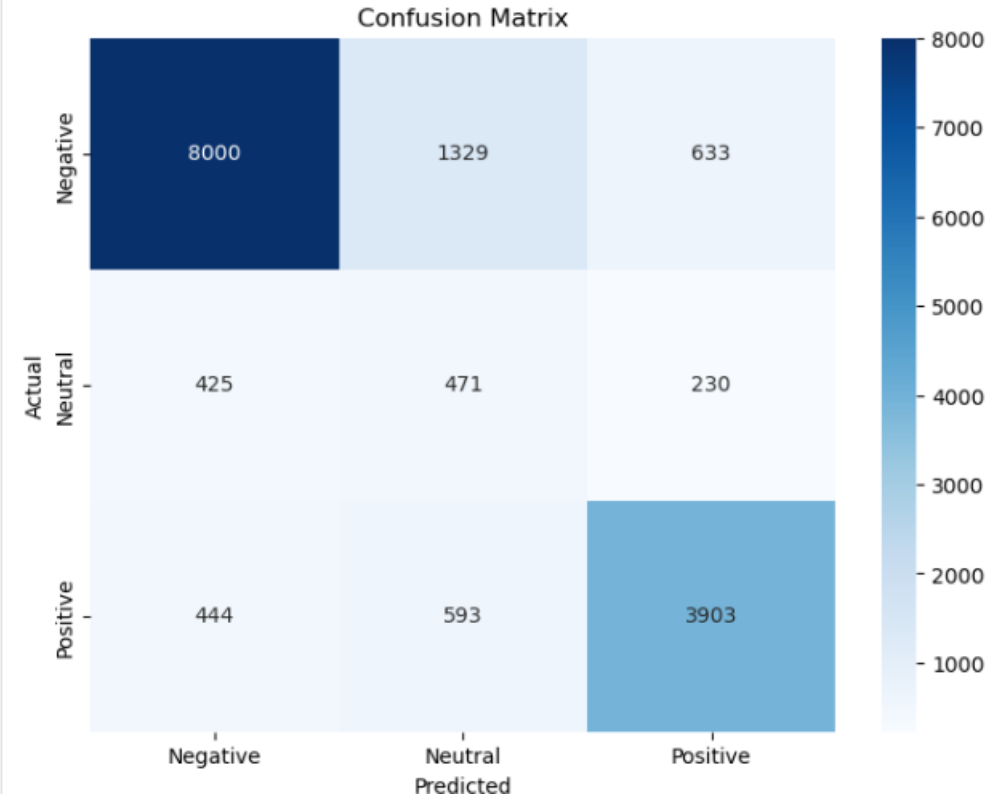
Logistic Regression Model + Class Weight 2

```
Best parameters: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
Overall Accuracy: 81.57%

Logistic Regression Model + GridSearchCV Report:
              precision    recall  f1-score   support

    Negative       0.89      0.88      0.88      9962
     Neutral       0.24      0.21      0.22      1126
    Positive       0.79      0.83      0.81      4940

    accuracy                           0.82     16028
   macro avg       0.64      0.64      0.64     16028
weighted avg       0.81      0.82      0.81     16028
```



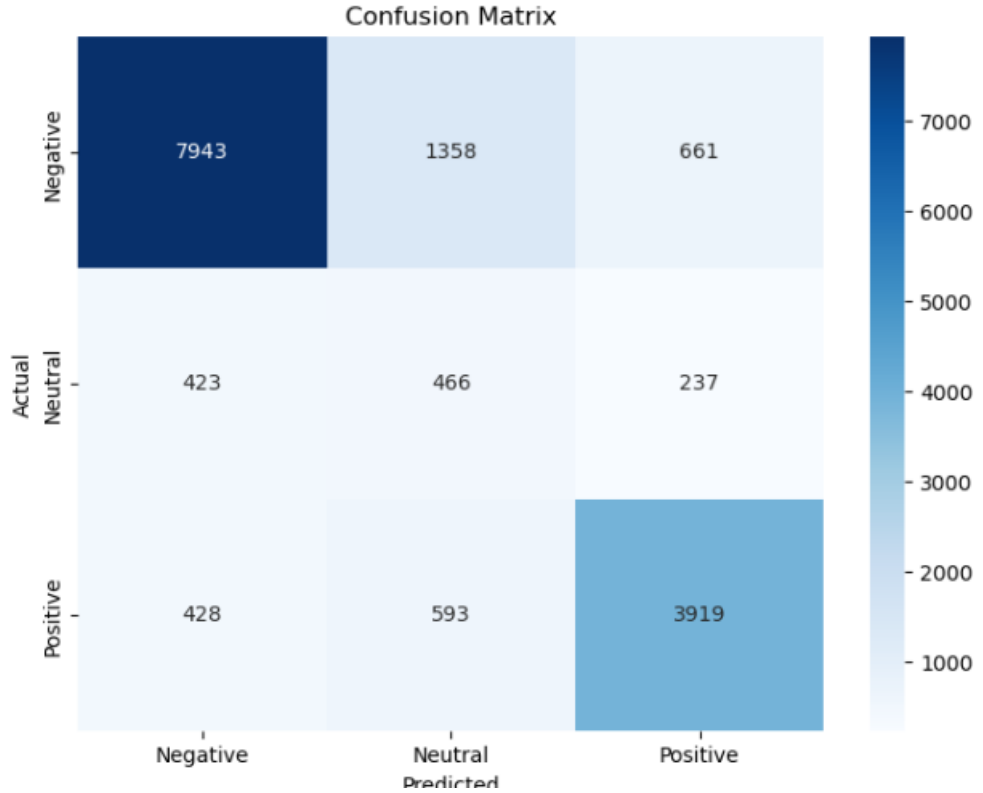
Logistic Regression Model + GridSearchCV

```
Overall Accuracy: 77.20%

Logistic regression model + Feature extraction optimization report:
              precision    recall  f1-score   support

    Negative       0.90      0.80      0.85      9962
     Neutral       0.20      0.42      0.27      1126
    Positive       0.82      0.79      0.80      4940

    accuracy                           0.77     16028
   macro avg       0.64      0.67      0.64     16028
weighted avg       0.83      0.77      0.79     16028
```



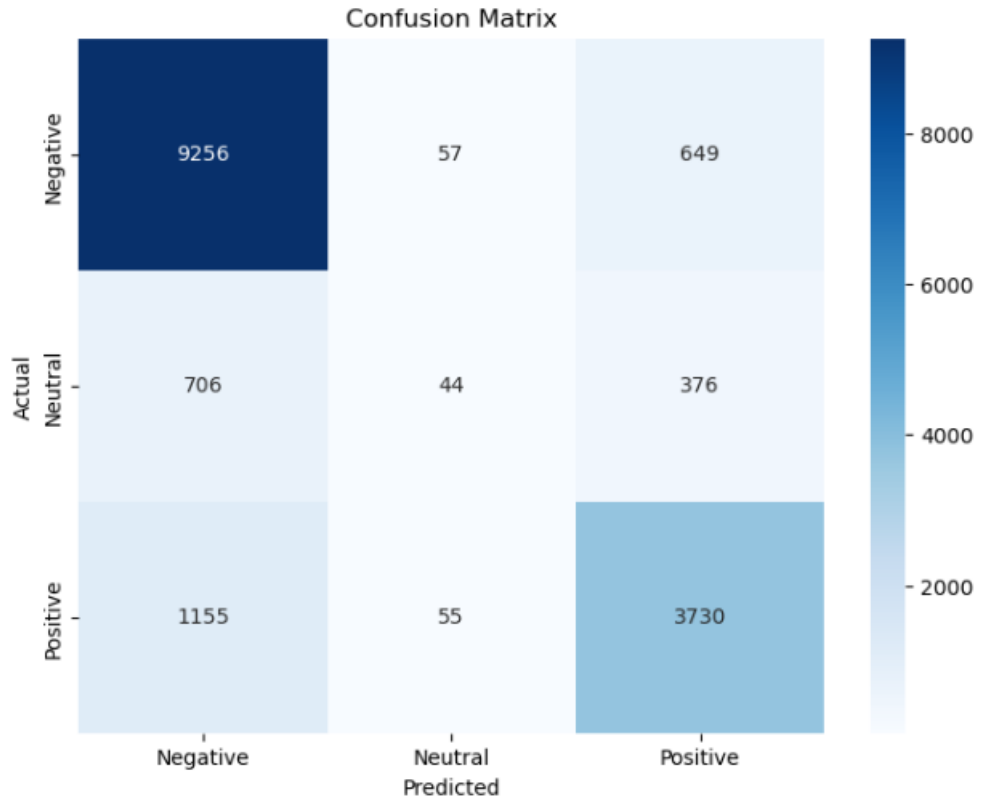
Logistic regression model + Feature extraction optimization

```
Overall Accuracy: 76.92%

Logistic regression model + Feature extraction optimization report2:
              precision    recall  f1-score   support

    Negative       0.90      0.80      0.85      9962
     Neutral       0.19      0.41      0.26      1126
    Positive       0.81      0.79      0.80      4940

    accuracy                           0.77     16028
   macro avg       0.64      0.67      0.64     16028
weighted avg       0.83      0.77      0.79     16028
```



Logistic regression model + Feature extraction optimization2

```
Overall Accuracy: 81.30%

Ensemble Learning report(Logistic regression, Random forests and Gradient boosting):
              precision    recall  f1-score   support

    Negative       0.83      0.93      0.88      9962
     Neutral       0.28      0.04      0.07      1126
    Positive       0.78      0.76      0.77      4940

    accuracy                           0.81     16028
   macro avg       0.63      0.57      0.57     16028
weighted avg       0.78      0.81      0.79     16028
```

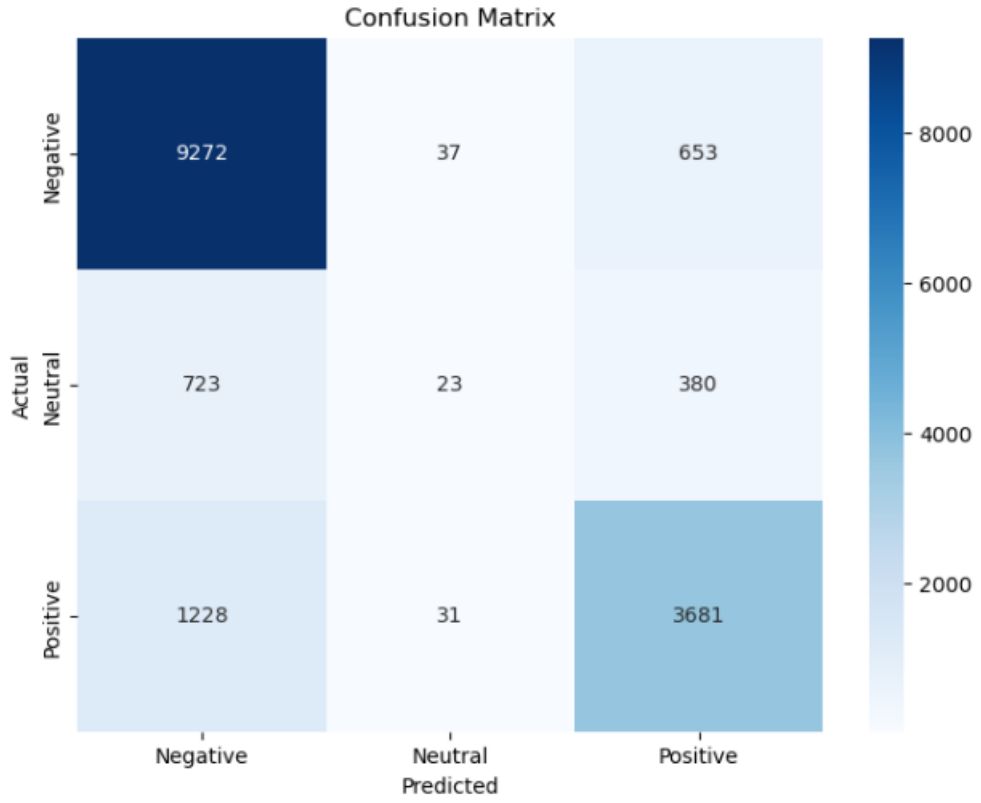


Ensemble Learning (Logistic regression, Random forests and Gradient boosting)

Overall Accuracy: 80.96%

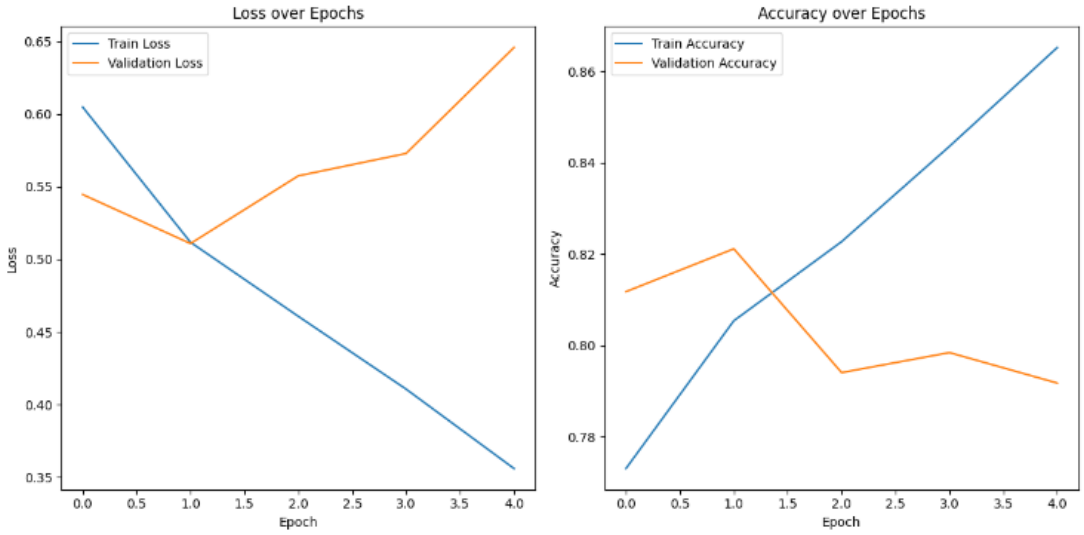
Ensemble Learning report2(Random forests, Gradient boosting and XGBoost):

	precision	recall	f1-score	support
Negative	0.83	0.93	0.88	9962
Neutral	0.25	0.02	0.04	1126
Positive	0.78	0.75	0.76	4940
accuracy			0.81	16028
macro avg	0.62	0.57	0.56	16028
weighted avg	0.77	0.81	0.78	16028



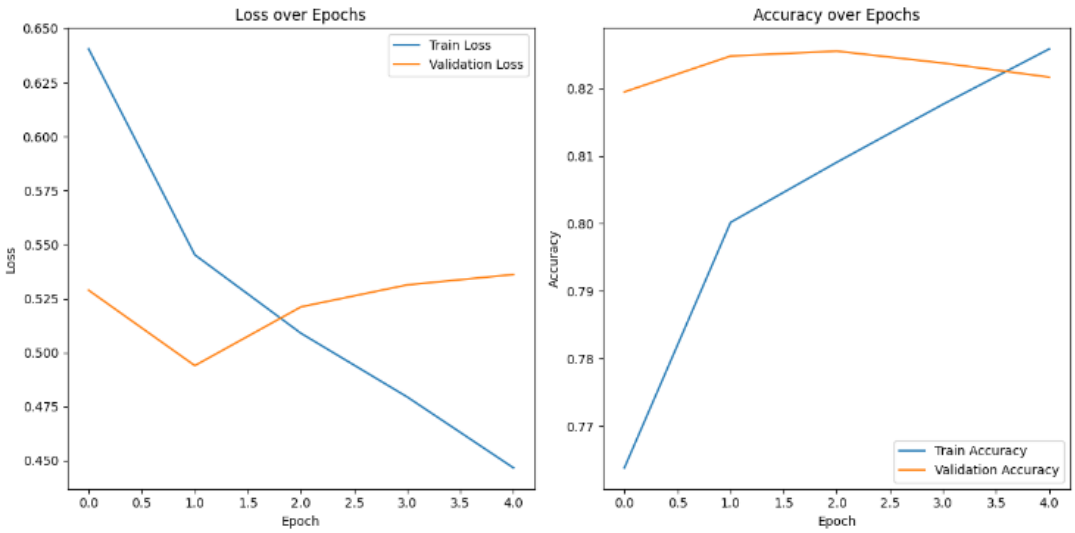
Ensemble Learning (Random forests, Gradient boosting and XGBoost)

```
Overall Accuracy: 79.83%
LSTM Model Report:
              precision    recall  f1-score   support

    Negative       0.88      0.87      0.87      9962
     Neutral       0.18      0.13      0.15      1126
    Positive       0.75      0.82      0.78      4940

    accuracy                           0.80     16028
   macro avg       0.60      0.60      0.60     16028
weighted avg       0.79      0.80      0.79     16028
```



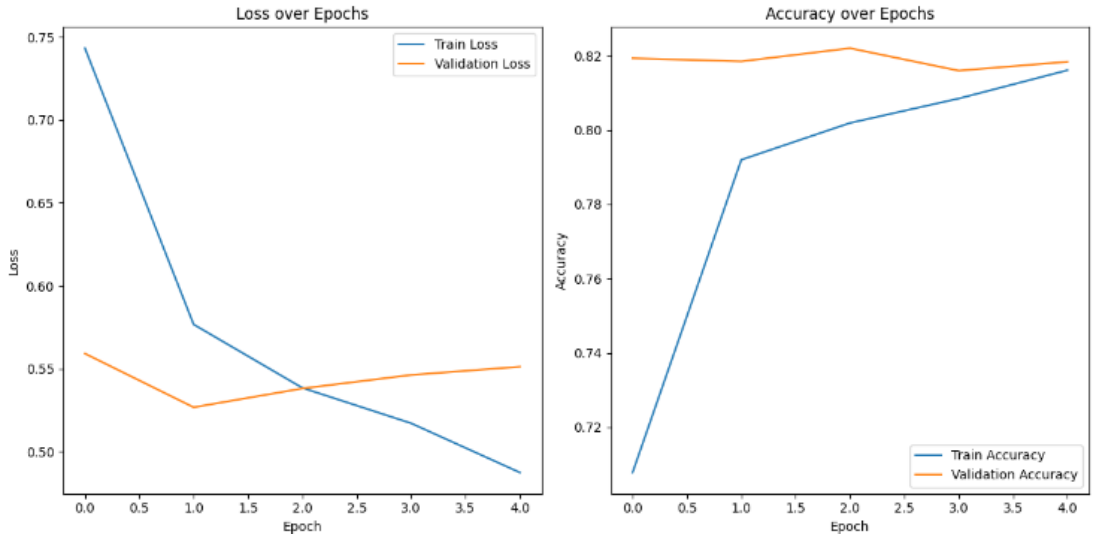
LSTM Model

```
Overall Accuracy:  82.52%
LSTM Model + Dropout Report:
              precision    recall  f1-score   support

    Negative       0.85      0.92      0.88      9962
     Neutral       0.30      0.02      0.05      1126
    Positive       0.78      0.81      0.80      4940

    accuracy                           0.83     16028
   macro avg       0.64      0.59      0.58     16028
weighted avg       0.79      0.83      0.80     16028
```



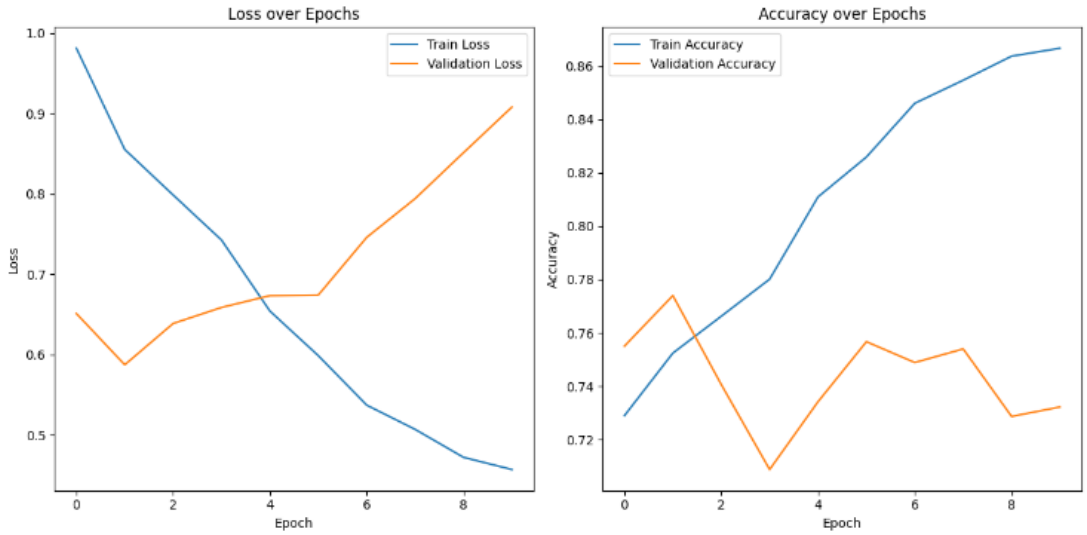
LSTM Model + Dropout

```
Overall Accuracy:  81.85%
LSTM Model + Dropout and BatchNormalization Report:
              precision    recall  f1-score   support

    Negative       0.86      0.90      0.88      9962
     Neutral       0.50      0.00      0.01      1126
    Positive       0.75      0.83      0.79      4940

    accuracy                           0.82     16028
   macro avg       0.70      0.58      0.56     16028
weighted avg       0.80      0.82      0.79     16028
```



LSTM Model + Dropout and BatchNormalization

```
Overall Accuracy:  74.15%
Bidirectional LSTM model + Class weights Report:
              precision    recall  f1-score   support

    Negative       0.88      0.77      0.82      9962
     Neutral       0.17      0.36      0.23      1126
    Positive       0.78      0.76      0.77      4940

    accuracy                           0.74     16028
   macro avg       0.61      0.63      0.61     16028
weighted avg       0.80      0.74      0.76     16028
```



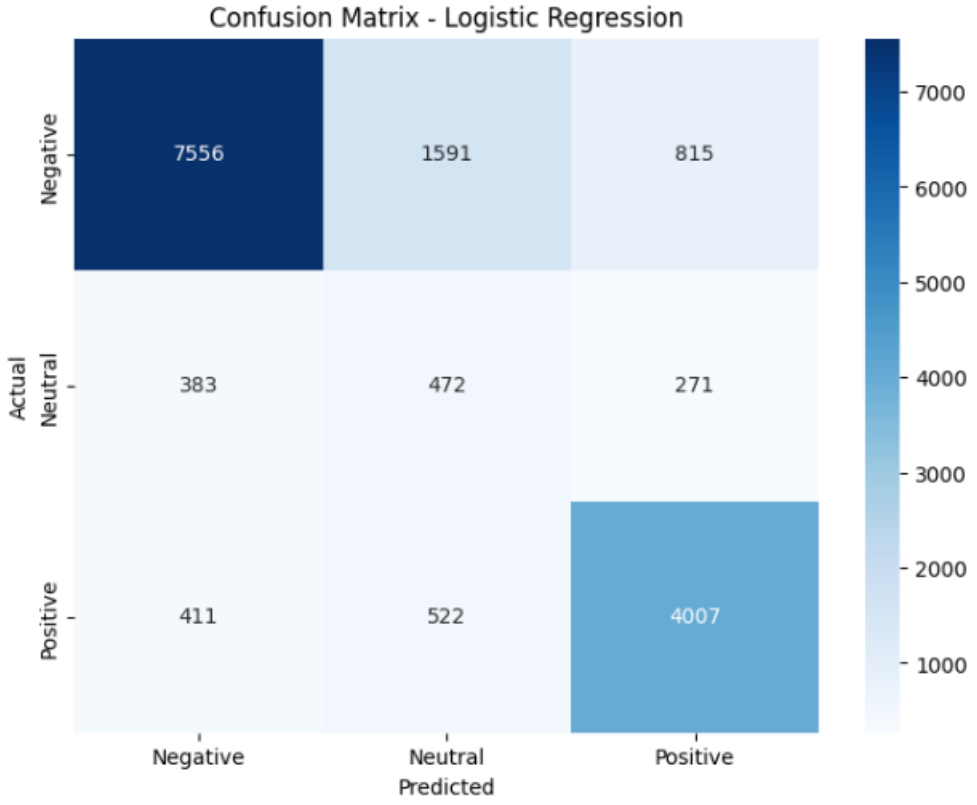
Bidirectional LSTM model + Class weights

BERT in sentiment analysis models

Overall Accuracy: 75.09%

Logistic Regression Model Report:

	precision	recall	f1-score	support
Negative	0.90	0.76	0.83	9962
Neutral	0.18	0.42	0.25	1126
Positive	0.79	0.81	0.80	4940
accuracy			0.75	16028
macro avg	0.62	0.66	0.63	16028
weighted avg	0.82	0.75	0.78	16028

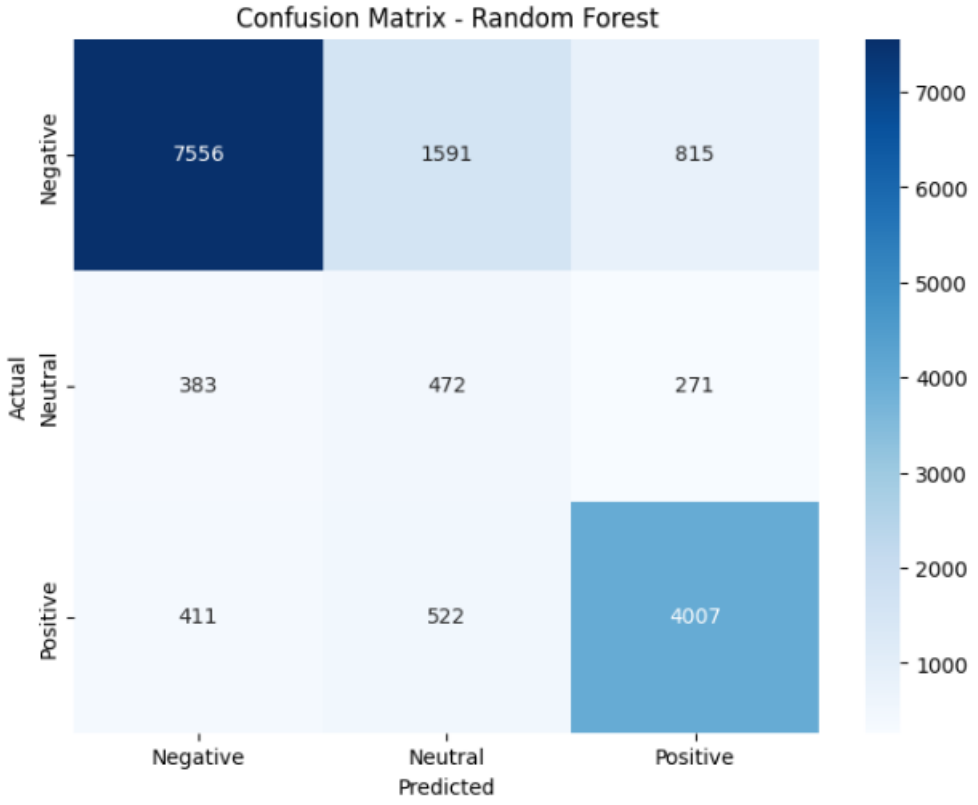


Logistic Regression Model

Overall Accuracy: 75.09%

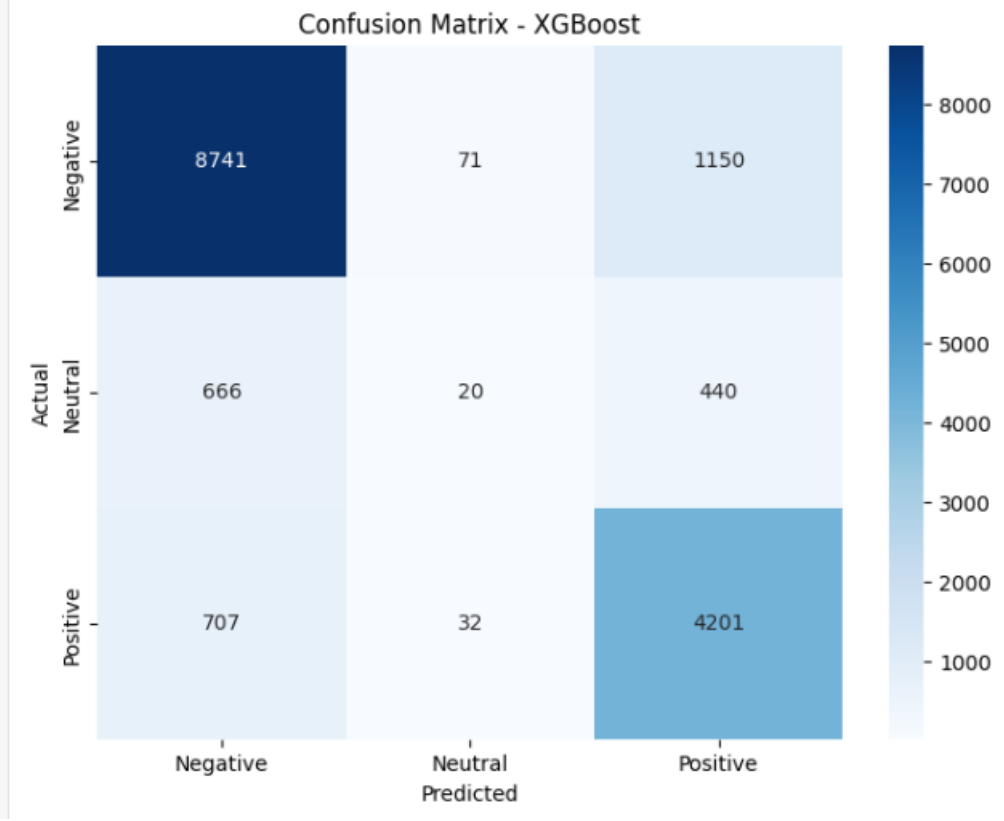
Random Forest Model + Class Weight Report:

	precision	recall	f1-score	support
Negative	0.90	0.76	0.83	9962
Neutral	0.18	0.42	0.25	1126
Positive	0.79	0.81	0.80	4940
accuracy			0.75	16028
macro avg	0.62	0.66	0.63	16028
weighted avg	0.82	0.75	0.78	16028



Random Forest Model

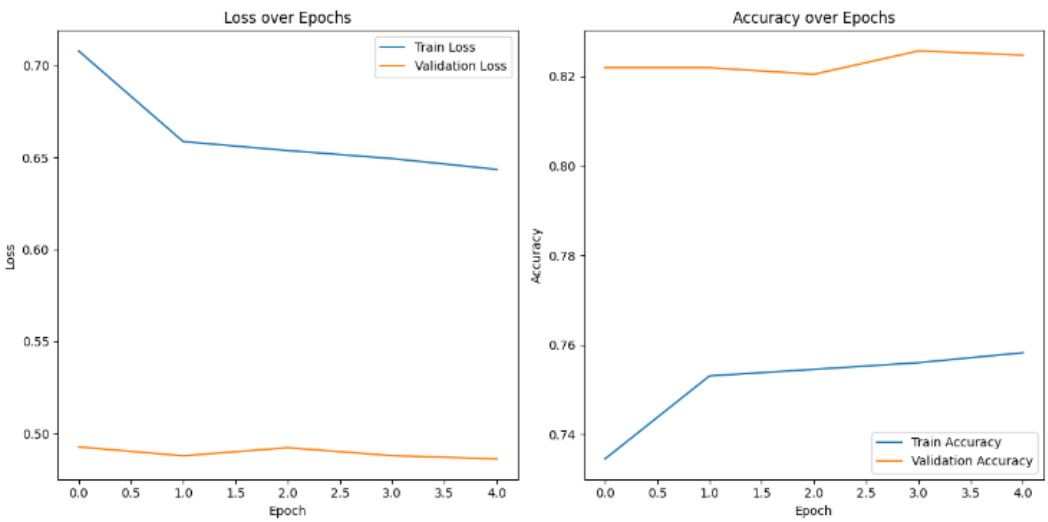
```
Overall Accuracy: 80.87%

XGBoost Model Report:
              precision    recall  f1-score   support

    Negative       0.86      0.88      0.87      9962
     Neutral       0.16      0.02      0.03      1126
    Positive       0.73      0.85      0.78      4940

    accuracy                           0.81     16028
   macro avg       0.58      0.58      0.56     16028
weighted avg       0.77      0.81      0.78     16028
```



XGBoost Model

```
Overall Accuracy: 82.63%

LSTM Model + Dropout and BatchNormalization Report:
              precision    recall  f1-score   support

    Negative       0.85      0.92      0.88      9962
     Neutral       0.00      0.00      0.00      1126
    Positive       0.77      0.84      0.80      4940

    accuracy                           0.83     16028
   macro avg       0.54      0.58      0.56     16028
weighted avg       0.77      0.83      0.80     16028
```

LSTM Model

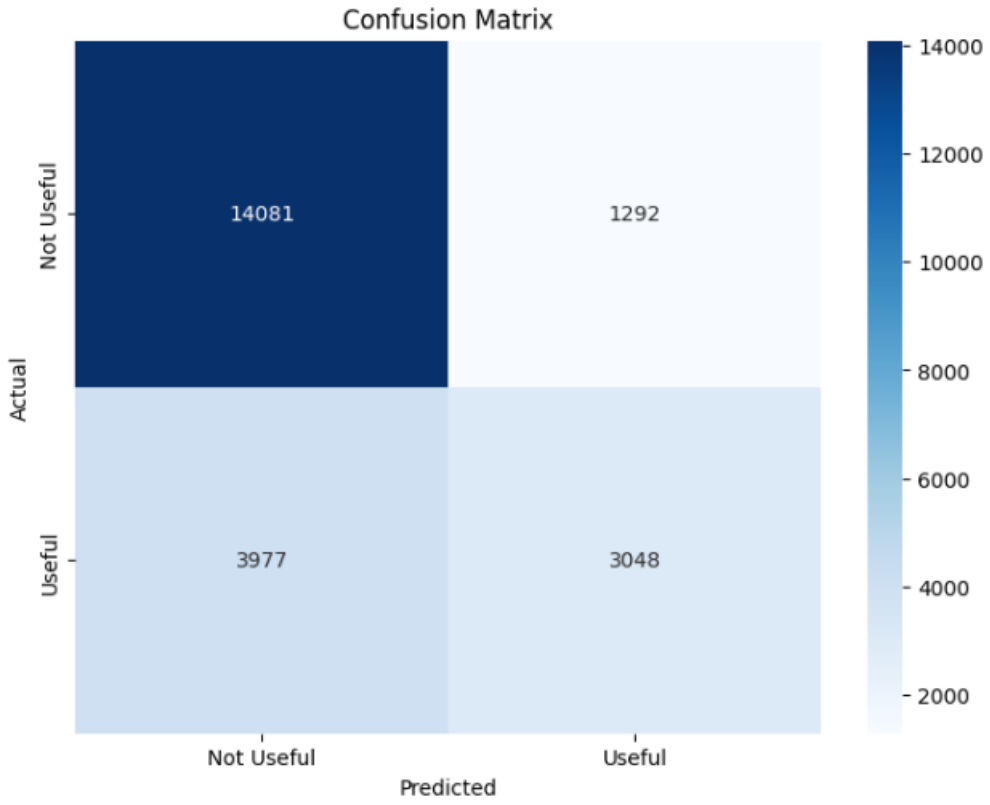


LSTM Model Loss and Accuracy Curve

TFIDF in usefulness score models

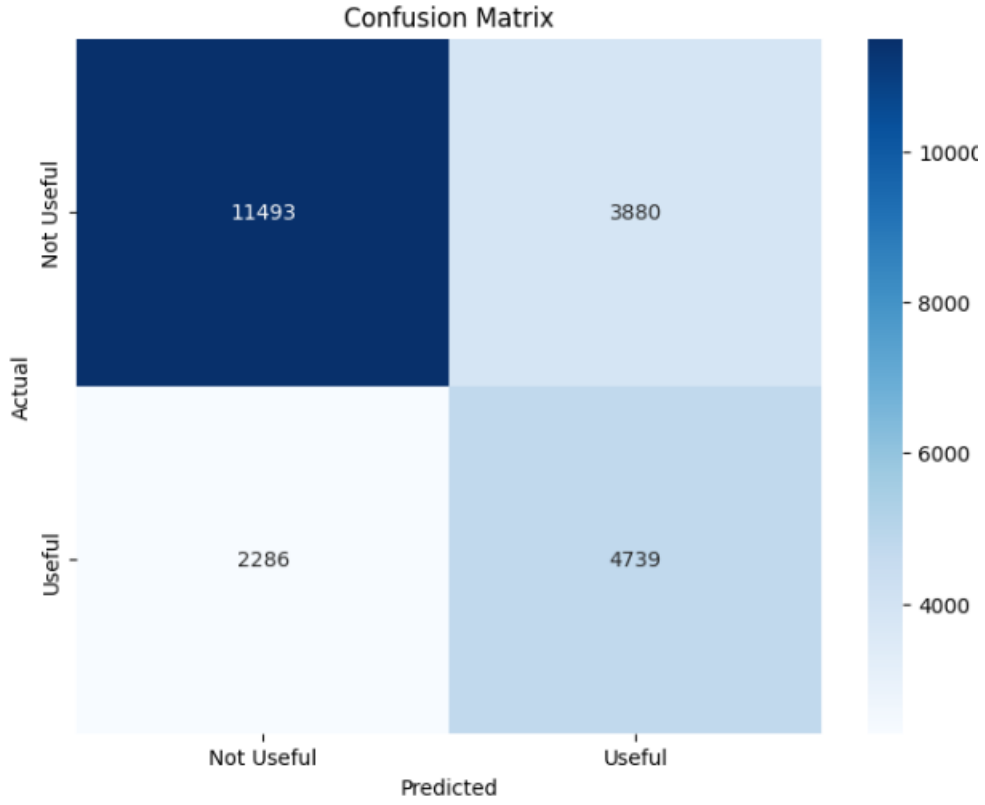
Overall Accuracy: 76.48%
Logistic Regression Model Report:

	precision	recall	f1-score	support
0	0.78	0.92	0.84	15373
1	0.70	0.43	0.54	7025
accuracy			0.76	22398
macro avg	0.74	0.67	0.69	22398
weighted avg	0.76	0.76	0.75	22398



Logistic Regression Model

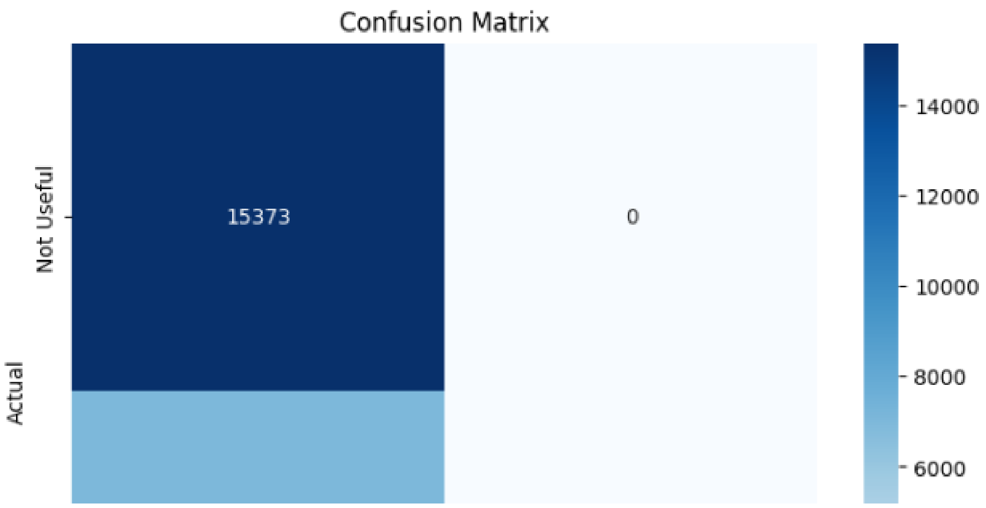
```
Overall Accuracy:  72.47%
Logistic Regression Model + Class Weight Report:
              precision    recall  f1-score   support

           0       0.83      0.75      0.79     15373
           1       0.55      0.67      0.61      7025

    accuracy                           0.72     22398
   macro avg       0.69      0.71      0.70     22398
weighted avg       0.74      0.72      0.73     22398
```



Logistic Regression Model + Class Weight

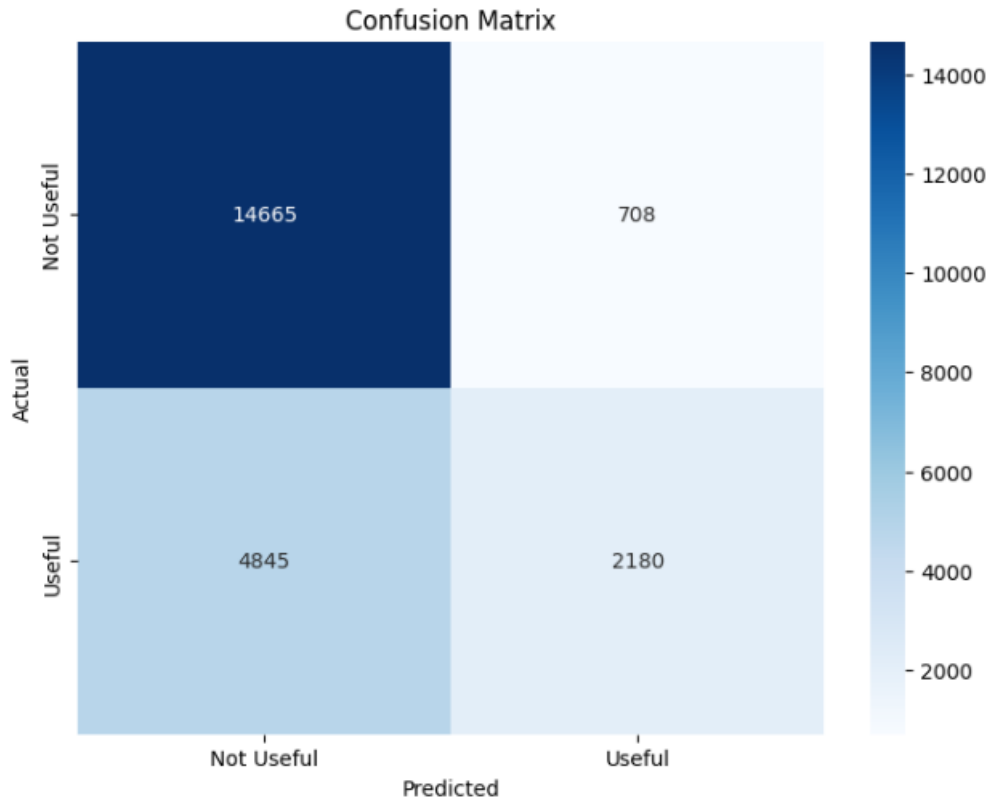
```
Overall Accuracy:  68.64%
Logistic Regression Model + Increase Iterations Report:
              precision    recall  f1-score   support

           0       0.69      1.00      0.81     15373
           1       0.00      0.00      0.00      7025

    accuracy                           0.69     22398
   macro avg       0.34      0.50      0.41     22398
weighted avg       0.47      0.69      0.56     22398
```

```
C:\Users\c22011387\OneDrive - Cardiff University\Desktop\Data-analysis\myenv\Lib\site-pack
trics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being s
abels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\c22011387\OneDrive - Cardiff University\Desktop\Data-analysis\myenv\Lib\site-pack
trics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being s
abels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\c22011387\OneDrive - Cardiff University\Desktop\Data-analysis\myenv\Lib\site-pack
trics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being s
abels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



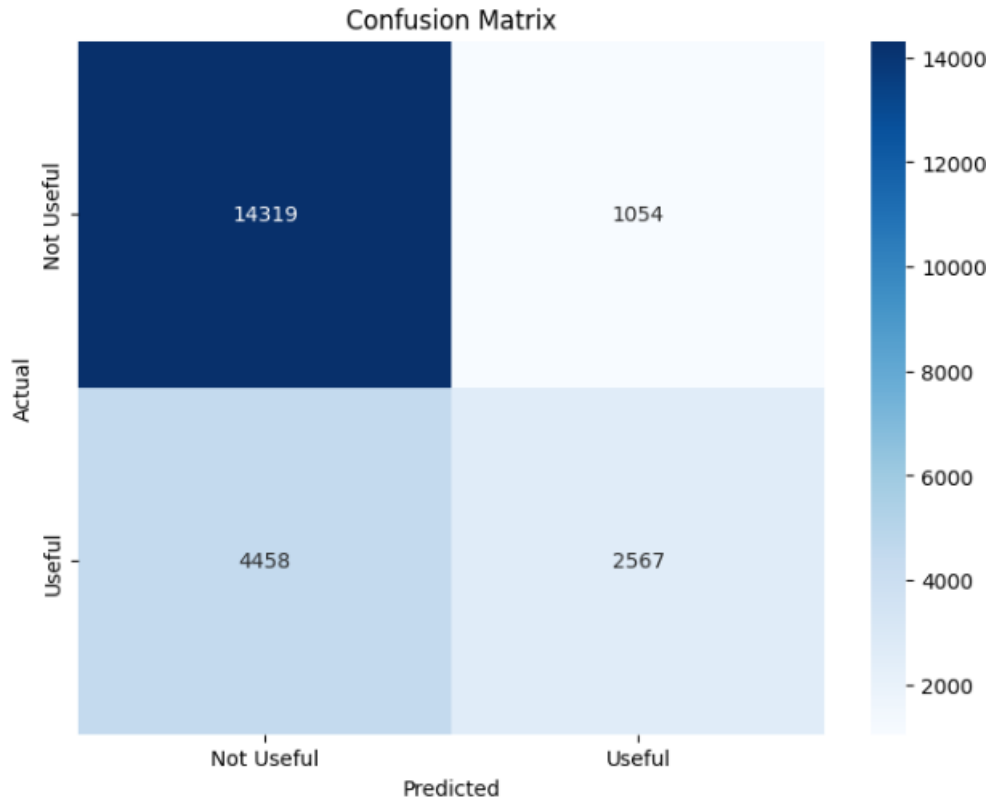
Logistic Regression Model + Increase Iterations

```
Overall Accuracy:  75.21%
Random Forest Model Report:
              precision    recall  f1-score   support

           0       0.75      0.95      0.84     15373
           1       0.75      0.31      0.44      7025

    accuracy                           0.75     22398
   macro avg       0.75      0.63      0.64     22398
weighted avg       0.75      0.75      0.72     22398
```



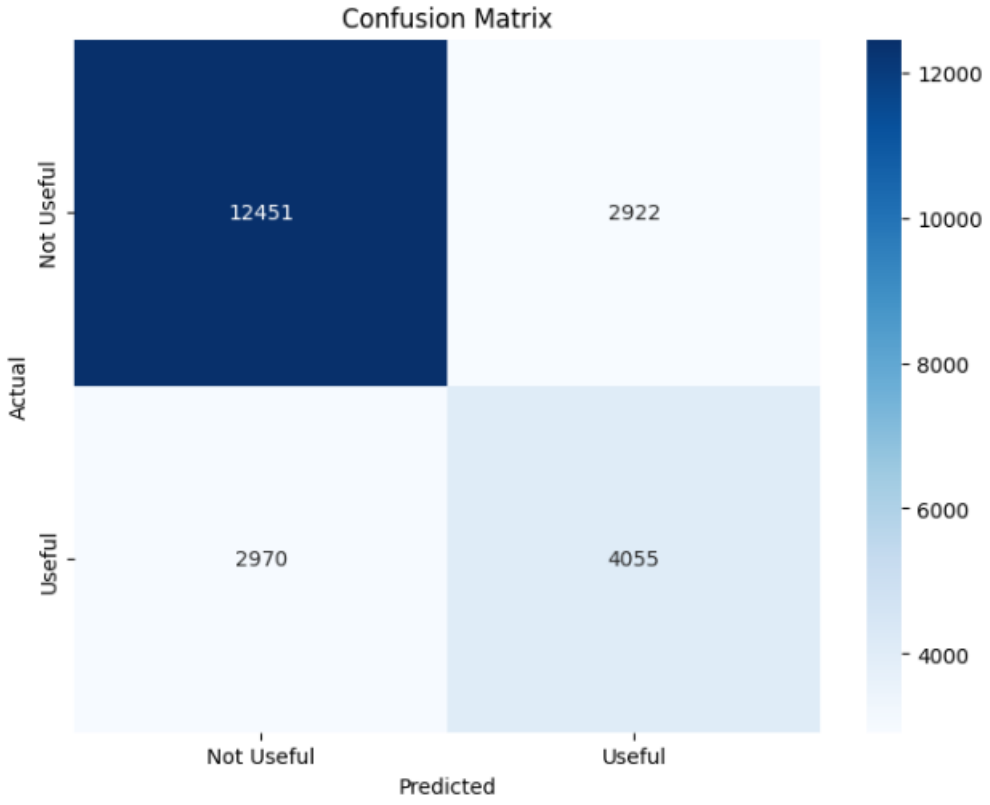
Random Forest Model

```
Overall Accuracy:  75.39%
Random Forest Model + Class Weight Report:
              precision    recall  f1-score   support

           0       0.76      0.93      0.84     15373
           1       0.71      0.37      0.48      7025

    accuracy                           0.75     22398
   macro avg       0.74      0.65      0.66     22398
weighted avg       0.75      0.75      0.73     22398
```



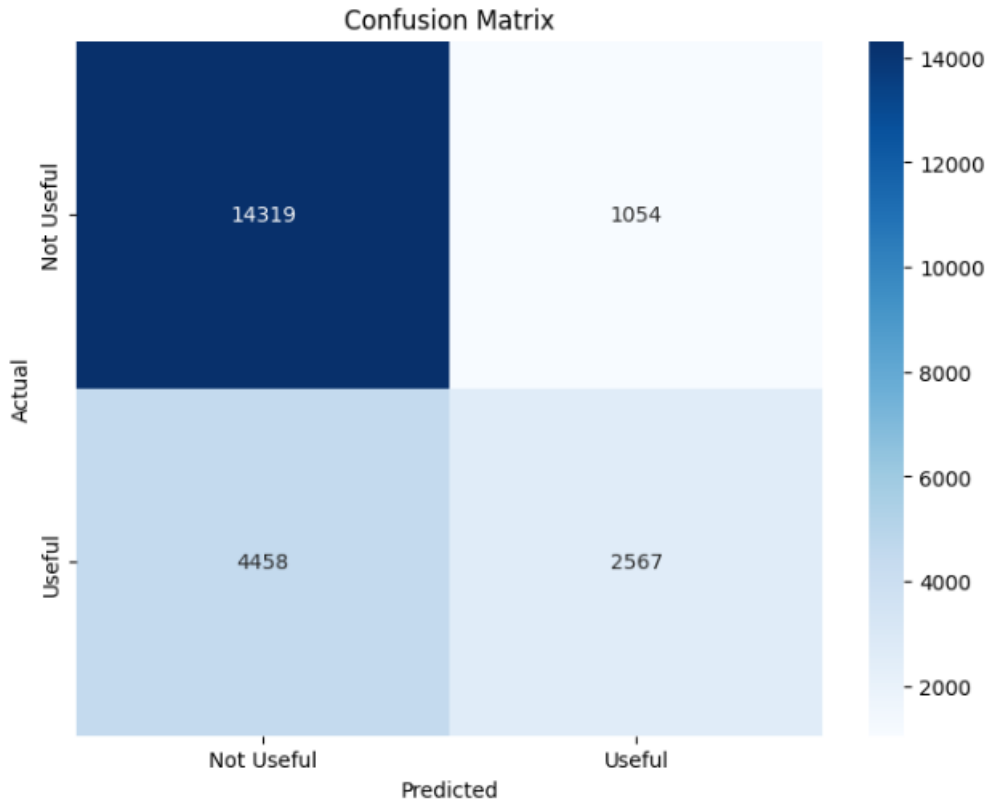
Random Forest Model + Class Weight

```
Best parameters (first stage): {'max_depth': 20, 'n_estimators': 100}
Best parameters (second stage): {'min_samples_split': 5}
Overall Accuracy:  73.69%
Random Forest Model + Hyperparameter Tuning with GridSearchCV Report:
              precision    recall  f1-score   support

           0       0.81      0.81      0.81     15373
           1       0.58      0.58      0.58      7025

    accuracy                           0.74     22398
   macro avg       0.69      0.69      0.69     22398
weighted avg       0.74      0.74      0.74     22398
```



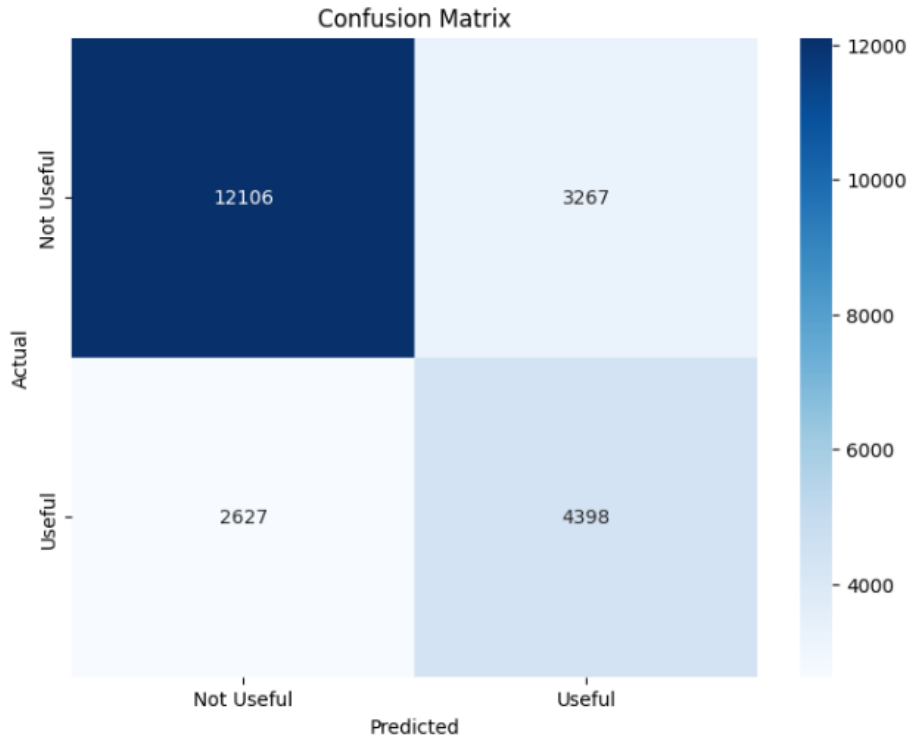
Random Forest Model + Hyperparameter Tuning with GridSearchCV

```
Overall Accuracy:  75.39%
XGBoost Model Report:
              precision    recall  f1-score   support

           0       0.76      0.93      0.84     15373
           1       0.71      0.37      0.48      7025

    accuracy                           0.75     22398
   macro avg       0.74      0.65      0.66     22398
weighted avg       0.75      0.75      0.73     22398
```



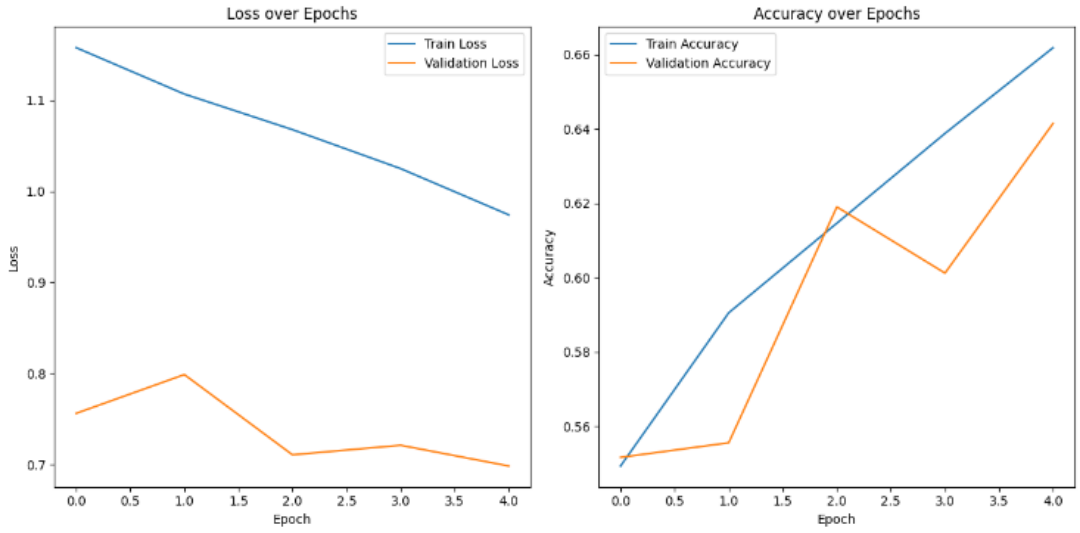
XGBoost Model

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50, 'scale_pos_weight': 2.1980466
63049376}
Overall Accuracy:  73.69%
XGBoost Model + Tuning with GridSearchCV Report:
              precision    recall  f1-score   support

           0       0.82      0.79      0.80     15373
           1       0.57      0.63      0.60      7025

    accuracy                           0.74     22398
   macro avg       0.70      0.71      0.70     22398
weighted avg       0.74      0.74      0.74     22398
```



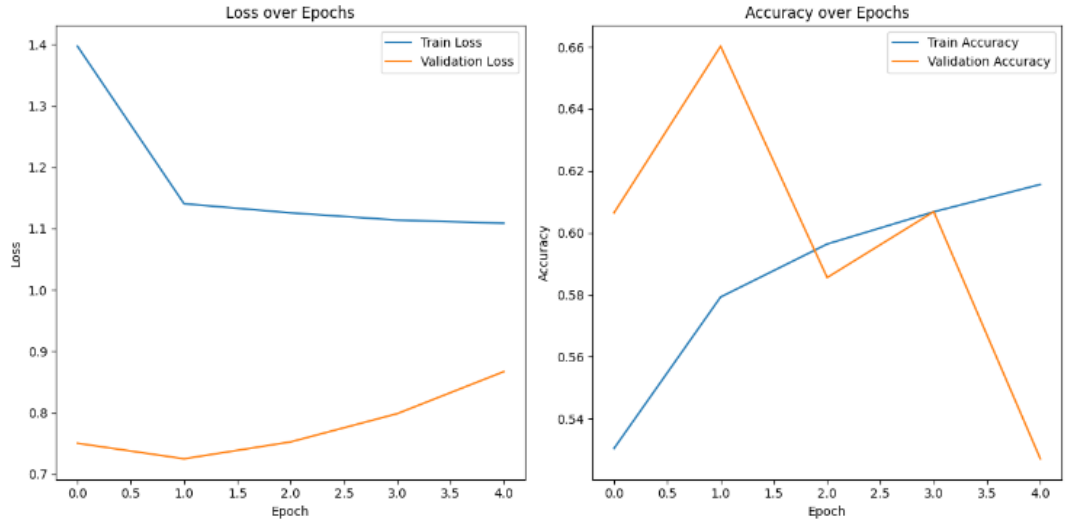
XGBoost Model + Tuning with GridSearchCV

```
Overall Accuracy:  64.15%
LSTM model Report:
              precision    recall  f1-score   support

           0       0.84      0.58      0.69     15373
           1       0.46      0.77      0.57      7025

    accuracy                           0.64     22398
   macro avg       0.65      0.68      0.63     22398
weighted avg       0.72      0.64      0.65     22398
```



LSTM model

```
Overall Accuracy:  66.02%
LSTM model Report:
              precision    recall  f1-score   support

           0       0.86      0.60      0.71     15373
           1       0.48      0.79      0.59      7025

    accuracy                           0.66     22398
   macro avg       0.67      0.70      0.65     22398
weighted avg       0.74      0.66      0.67     22398
```

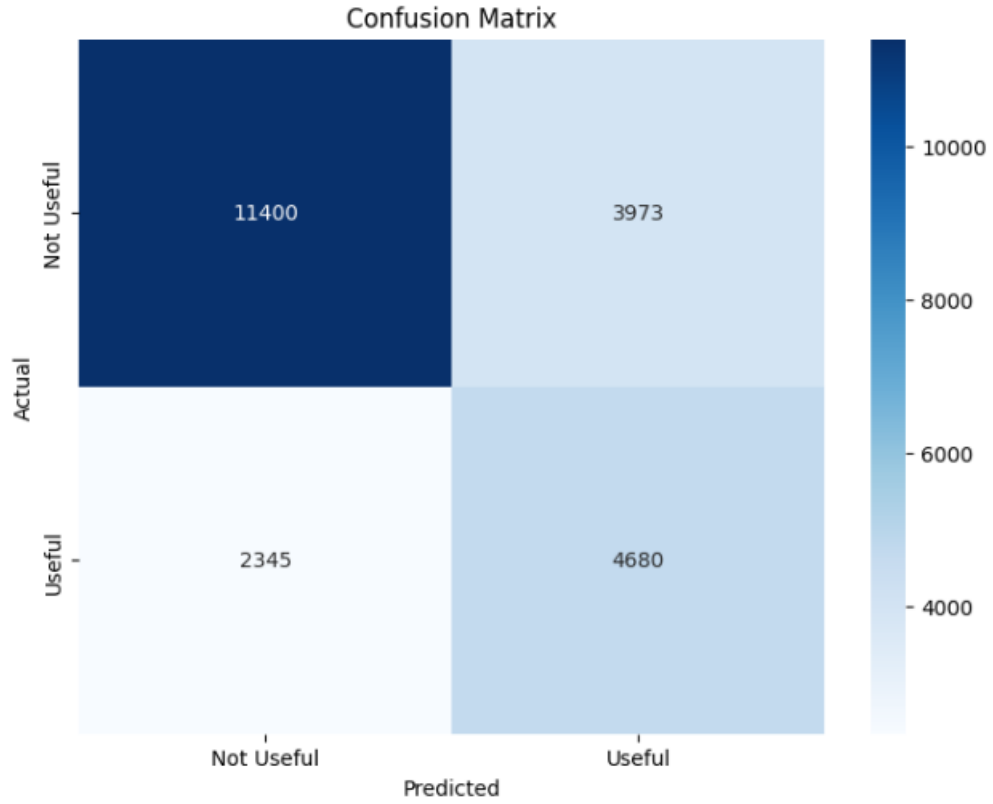


LSTM Model, Increased units and L2 regularization

BERT in usefulness score models

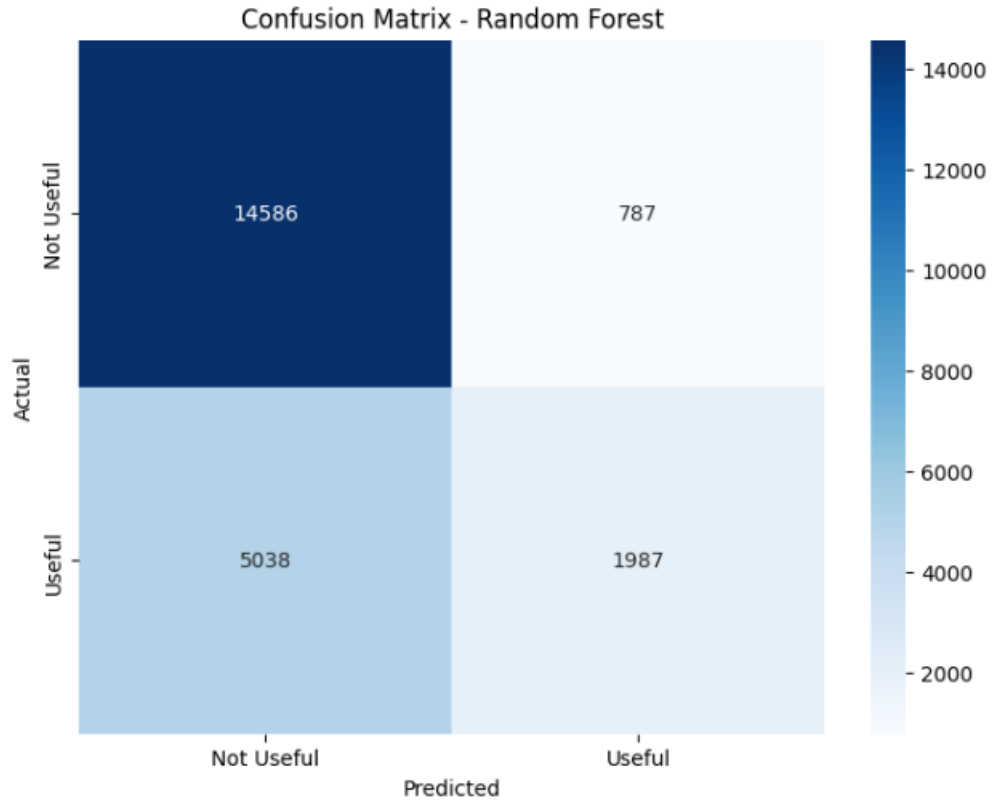
Overall Accuracy: 71.79%
Logistic Regression Model Report:

	precision	recall	f1-score	support
0	0.83	0.74	0.78	15373
1	0.54	0.67	0.60	7025
accuracy			0.72	22398
macro avg	0.69	0.70	0.69	22398
weighted avg	0.74	0.72	0.72	22398



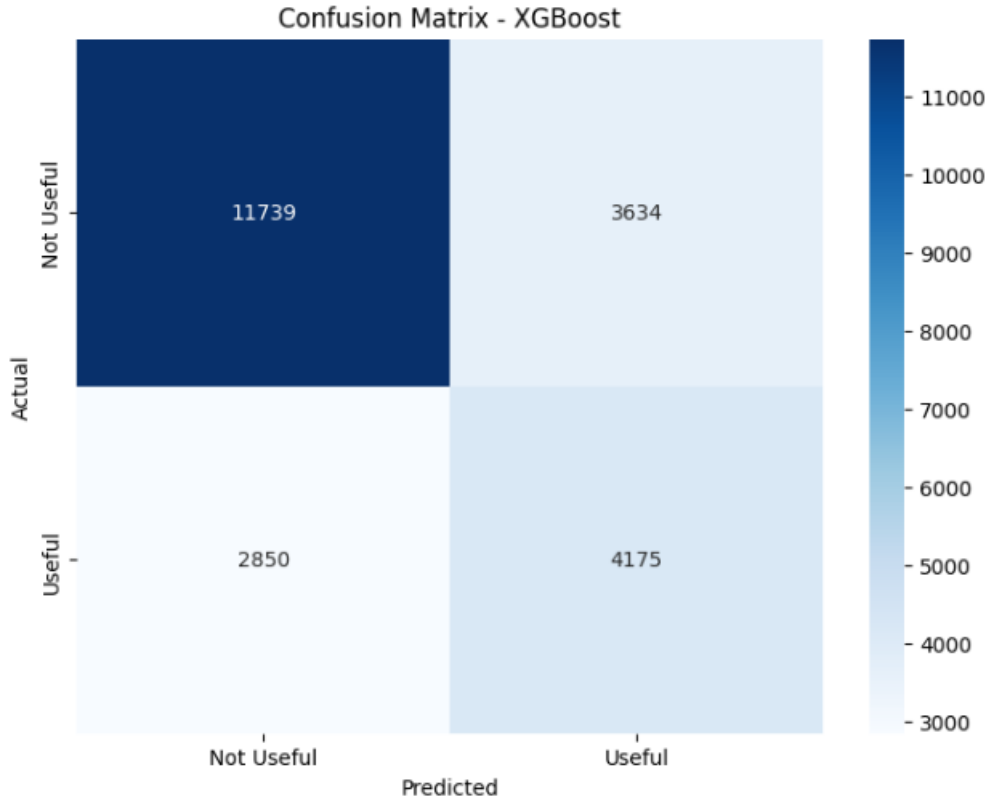
Logistic Regression Model

```
Overall Accuracy (Random Forest):  73.99%
Random Forest Model Report:
              precision    recall  f1-score   support

           0       0.74      0.95      0.83     15373
           1       0.72      0.28      0.41      7025

    accuracy                           0.74     22398
   macro avg       0.73      0.62      0.62     22398
weighted avg       0.73      0.74      0.70     22398
```



Random Forest Model

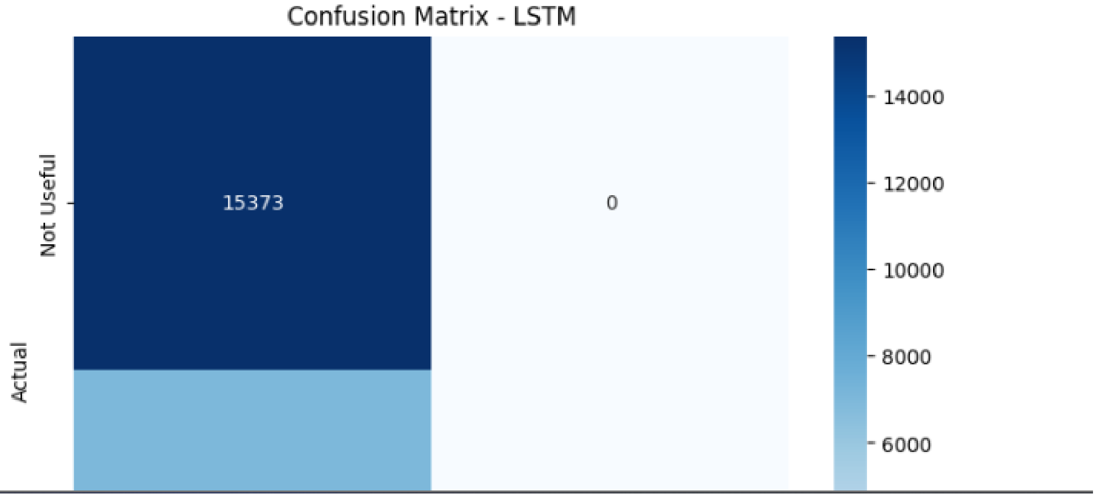
```
Overall Accuracy (XGBoost):  71.05%
XGBoost Model Report:
              precision    recall  f1-score   support

           0       0.80      0.76      0.78     15373
           1       0.53      0.59      0.56      7025

    accuracy                           0.71     22398
   macro avg       0.67      0.68      0.67     22398
weighted avg       0.72      0.71      0.71     22398
```



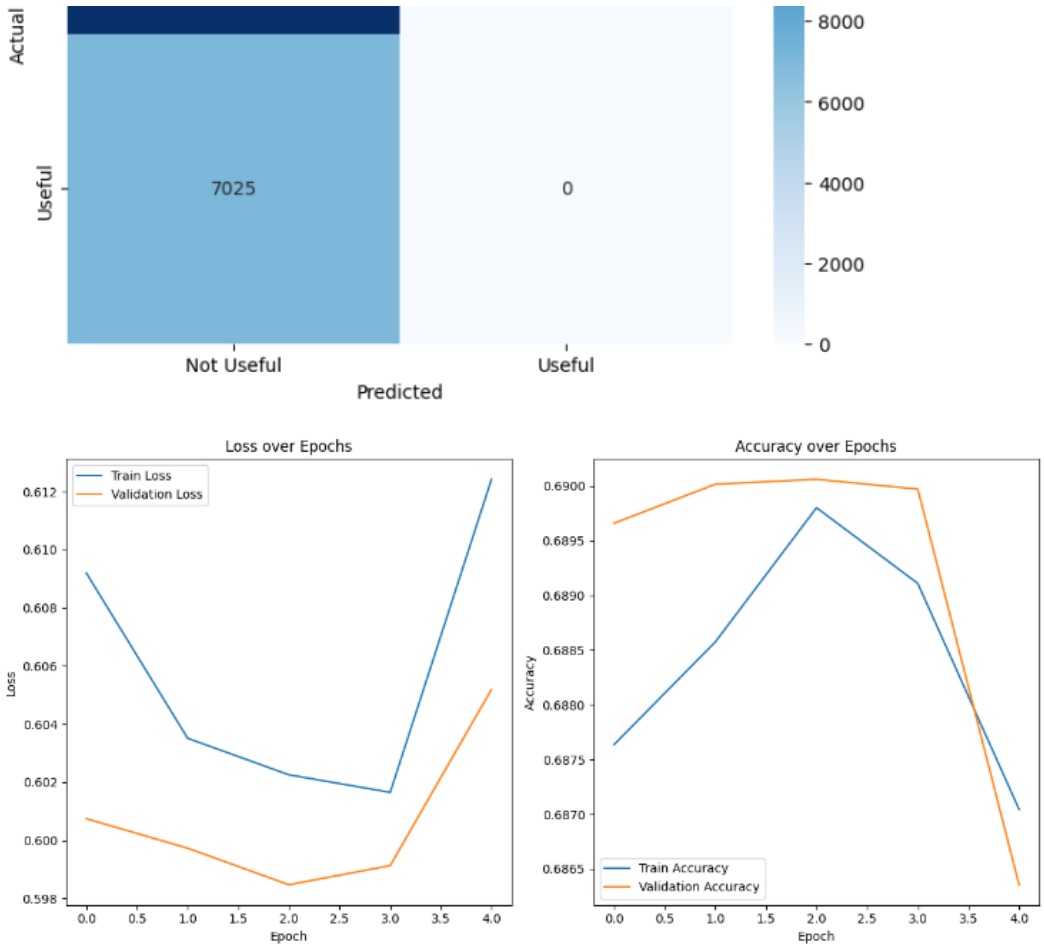
XGBoost Model

```
Overall Accuracy (LSTM):  68.64%
LSTM Model Report:
              precision    recall  f1-score   support

           0       0.69      1.00      0.81     15373
           1       0.00      0.00      0.00      7025

    accuracy                           0.69     22398
   macro avg       0.34      0.50      0.41     22398
weighted avg       0.47      0.69      0.56     22398
```

```
C:\Users\c22011387\OneDrive - Cardiff University\Desktop\Data-analysis\myenv\Lib\site-packages\sklearn
trics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 i
abels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\c22011387\OneDrive - Cardiff University\Desktop\Data-analysis\myenv\Lib\site-packages\sklearn
trics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 i
abels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\c22011387\OneDrive - Cardiff University\Desktop\Data-analysis\myenv\Lib\site-packages\sklearn
trics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 i
abels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

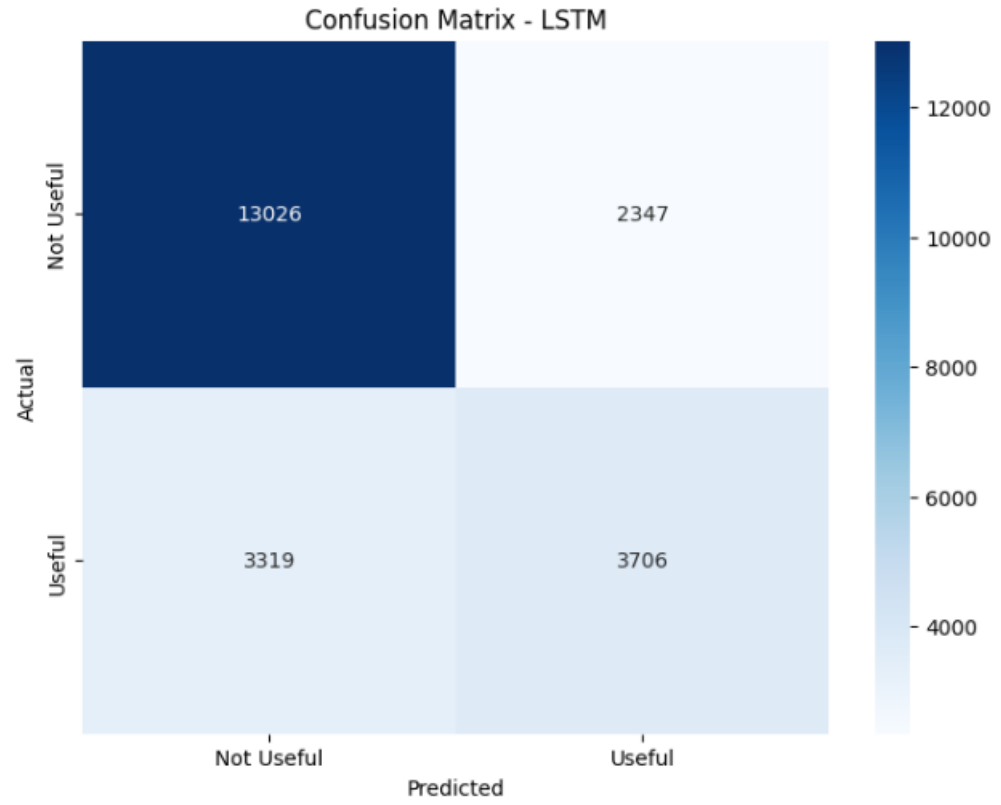


LSTM Model

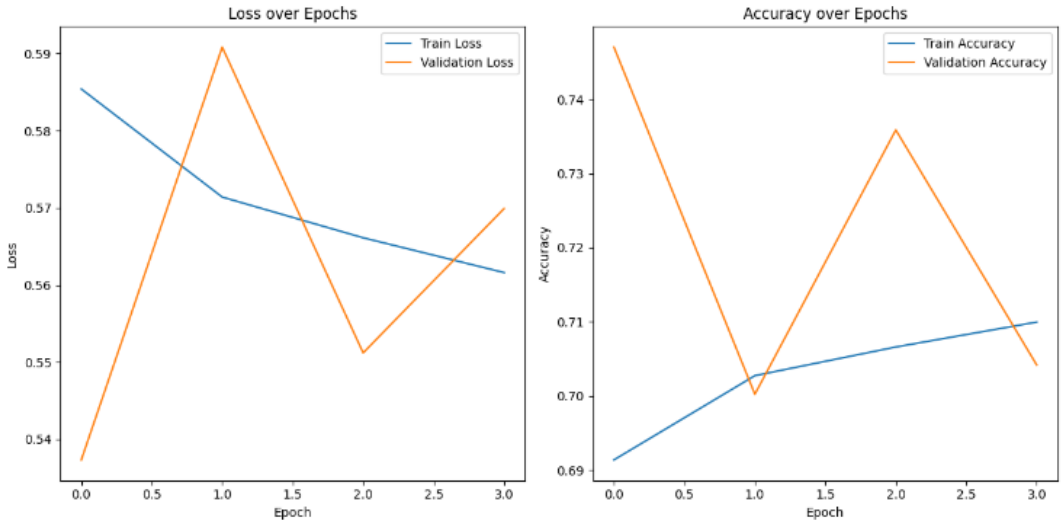

Actual
Useful
7025
0
8000
6000
4000
2000
0
Not Useful
Useful
Predicted
Loss over Epochs
Train Loss
Validation Loss
Loss
Epoch
Accuracy over Epochs
Accuracy
Train Accuracy
Validation Accuracy
Epoch

LSTM Model 2

```
Overall Accuracy (LSTM):  74.70%
LSTM Model Report:
              precision    recall  f1-score   support

           0       0.80      0.85      0.82     15373
           1       0.61      0.53      0.57      7025

    accuracy                           0.75     22398
   macro avg       0.70      0.69      0.69     22398
weighted avg       0.74      0.75      0.74     22398
```

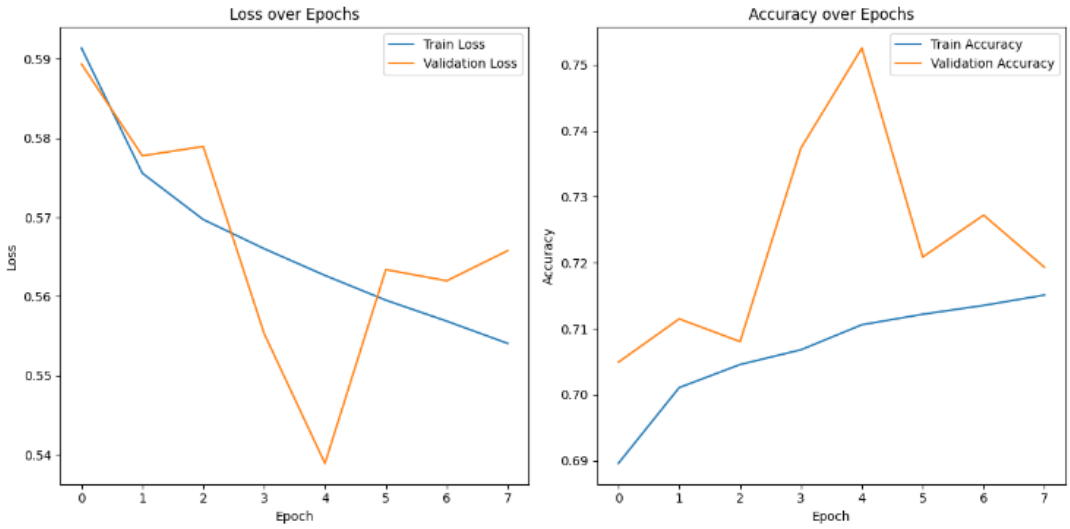


LSTM+Smote, Class weights Confusion Matrix



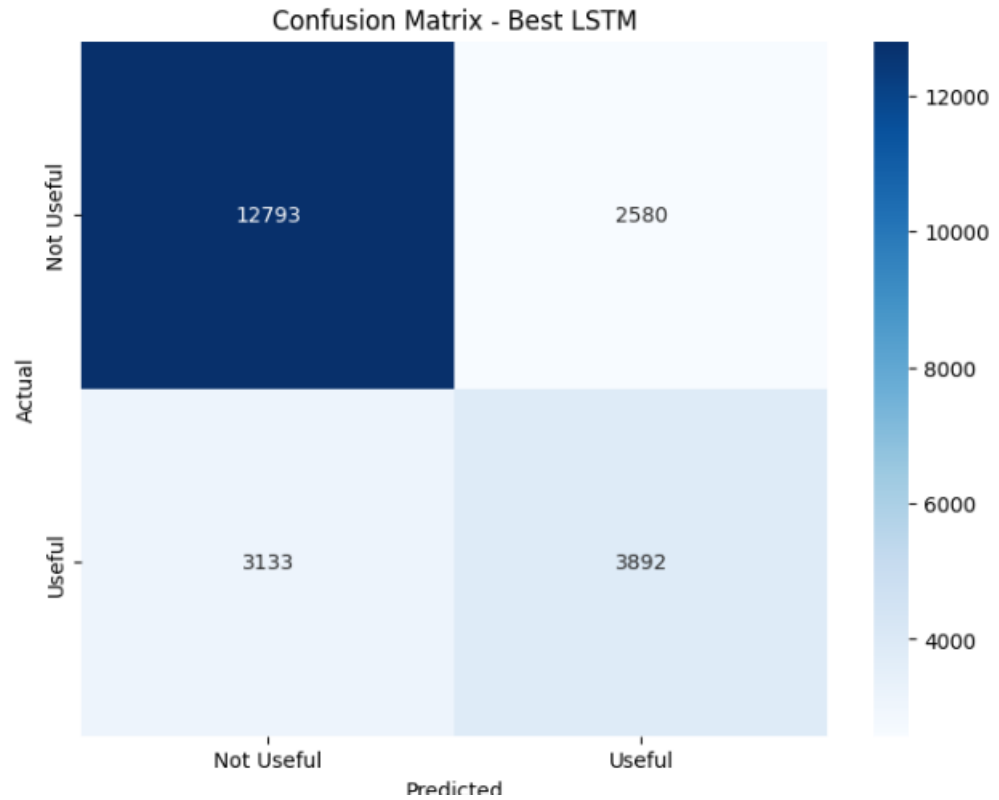
LSTM+Smote, Class weights Loss and Accuracy Curve

```
Accuracy: 0.7419
Best Parameters: {'dropout_rate': 0.5, 'units': 16}
Best Accuracy: 0.7525
```



LSTM+Smote, GridSearch Loss and Accuracy Curve

```
Overall Accuracy (Best LSTM):  74.49%
Best LSTM Model Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.82     15373
           1       0.60      0.55      0.58      7025

    accuracy                           0.74     22398
   macro avg       0.70      0.69      0.70     22398
weighted avg       0.74      0.74      0.74     22398
```



Best LSTM+Smote, GridSearch Confusion Matrix

```
Reviews where BERT Logistic Regression is wrong but TF-IDF Logistic Regression is correct:

Review: netflix i would have once got a  star review however this review is based on  things  the app is constantly trying to show me wh
at it thinks i should watch which buries content that i find only when other people tell me to look up a certain show  my kids live with
me  of the time now the app doesnt work on their smart devices because it thinks they dont live here now i have changed internet service
s it does the same to me at home and they want me to pay an account upgrade
True Sentiment: 0 (Label: Negative)
BERT Prediction: 1 (Label: Neutral)
TF-IDF Prediction: 0 (Label: Negative)

Review: sexual content
True Sentiment: 0 (Label: Negative)
BERT Prediction: 2 (Label: Positive)
TF-IDF Prediction: 0 (Label: Negative)

Review: i hope you fix the bug that logs me out every time i access the app with the error that im not part of the netflix household whe
n im the main owner  thank you
True Sentiment: 0 (Label: Negative)
BERT Prediction: 2 (Label: Positive)
TF-IDF Prediction: 0 (Label: Negative)

Review: i thought that all of the old and new movie is in the netflix
True Sentiment: 0 (Label: Negative)
BERT Prediction: 1 (Label: Neutral)
TF-IDF Prediction: 0 (Label: Negative)

Review: its so difficult to sign in
True Sentiment: 0 (Label: Negative)
BERT Prediction: 1 (Label: Neutral)
TF-IDF Prediction: 0 (Label: Negative)
```

Reviews where BERT Logistic Regression is wrong but TF-IDF Logistic Regression is correct

```
Reviews where BERT Random Forest is wrong but TF-IDF Random Forest is correct:

Review: sexual content
True Sentiment: 0 (Label: Negative)
BERT Random Forest Prediction: 2 (Label: Positive)
TF-IDF Random Forest Prediction: 0 (Label: Negative)

Review: very nice app personally i admire this app because no ads no buffer so smoothly runs  some companies  i wouldnt like to mention th
eir namescant give such experience like netflixat the end of the day i feel relaxed by watching netflix
True Sentiment: 2 (Label: Positive)
BERT Random Forest Prediction: 0 (Label: Negative)
TF-IDF Random Forest Prediction: 2 (Label: Positive)

Review: i do not liked it at all
True Sentiment: 0 (Label: Negative)
BERT Random Forest Prediction: 2 (Label: Positive)
TF-IDF Random Forest Prediction: 0 (Label: Negative)

Review: i hope you fix the bug that logs me out every time i access the app with the error that im not part of the netflix household when
im the main owner  thank you
True Sentiment: 0 (Label: Negative)
BERT Random Forest Prediction: 2 (Label: Positive)
TF-IDF Random Forest Prediction: 0 (Label: Negative)

Review:  pradeepd
True Sentiment: 0 (Label: Negative)
BERT Random Forest Prediction: 2 (Label: Positive)
TF-IDF Random Forest Prediction: 0 (Label: Negative)

Review: pro zionist app
True Sentiment: 0 (Label: Negative)
BERT Random Forest Prediction: 2 (Label: Positive)
TF-IDF Random Forest Prediction: 0 (Label: Negative)

Review: who am i think and awnser me
True Sentiment: 2 (Label: Positive)
BERT Random Forest Prediction: 0 (Label: Negative)
TF-IDF Random Forest Prediction: 2 (Label: Positive)
```

Reviews where BERT Random Forest is wrong but TF-IDF Random Forest is correct

```
Reviews where BERT XGBoost is wrong but TF-IDF XGBoost is correct:

Review: i do not liked it at all
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)

Review: i hope you fix the bug that logs me out every time i access the app with the error that im not part of the netflix household when
im the main owner  thank you
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)

Review:  pradeepd
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)

Review: pro zionist app
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)

Review: very very poor
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)

Review: net work isssue
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)

Review:  app
True Sentiment: 0 (Label: Negative)
BERT XGBoost Prediction: 2 (Label: Positive)
TF-IDF XGBoost Prediction: 0 (Label: Negative)
```

Reviews where BERT XGBoost is wrong but TF-IDF XGBoost is correct

```
Reviews where BERT LSTM is wrong but TF-IDF LSTM is correct:

Review: i do not liked it at all
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 2 (Label: Positive)
TF-IDF LSTM Prediction: 2 (Label: Positive)

Review: i thought that all of the old and new movie is in the netflix
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 2 (Label: Positive)
TF-IDF LSTM Prediction: 2 (Label: Positive)

Review: pro zionist app
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 2 (Label: Positive)

Review: who am i think and awnser me
True Sentiment: 2 (Label: Positive)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: login problem login atemet
True Sentiment: 0 (Label: Negative)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: so gooodddd
True Sentiment: 2 (Label: Positive)
BERT LSTM Prediction: 0 (Label: Negative)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: please challenge kar do
True Sentiment: 2 (Label: Positive)
BERT LSTM Prediction: 2 (Label: Positive)
TF-IDF LSTM Prediction: 0 (Label: Negative)

Review: jaana jeele apni jindagi
True Sentiment: 2 (Label: Positive)
```

Reviews where BERT LSTM is wrong but TF-IDF LSTM is correct