FACULTY OF COMPUTING AND INFORMATICS

BACHLEOR IN COMPUTER SCIENCE

SOCIAL MEDIA COMPUTING – CDS6344

TRIMESTER, Session 2024/2025

Project Title: Sentiment Analysis of Uber Customer Reviews

Prepared By:

Name	Student ID:
Leong Jia Yi	1211101506
Ter Zheng Bin	1211103705

Code Link:

Google Collab	Link
GitHub:	Link

Contents

1. Introduction

In today's digital age of mobility, ride-hailing apps have revolutionized transportation patterns by providing consumers with more convenience, increased accessibility, and instant connectivity. Amongst the various existing services, Uber has emerged as a leading global brand, serving millions of users across various geographic areas globally. At the same time, the expansion of user bases has been followed by a high increase in customer reviews and ratings, especially on sites like the Google Play Store. Reviews provide useful information on user satisfaction, complaints, and suggestions, thus making them a crucial tool for optimizing services using data-driven approaches.

However, the manual inspection of a large number of feedback entries is inefficient and not scalable. This situation highlights the importance of sentiment analysis, which is a **key natural language processing (NLP)** function. With the application of NLP practices and machine learning techniques, it is possible to systematically classify customers' sentiments as **positive**, **neutral**, or **negative**, and thus derive valuable insights from unstructured text reviews.

This project aims to analyze over 12,000 anonymized Uber reviews collected from the Google Play Store. It implements models including Logistic Regression, Support Vector Machine (SVM), Random Forest, and Bidirectional Long Short-Term Memory (BiLSTM). Additionally, it incorporates Transformer-based models, particularly *DistilBERT*, to explore how pre-trained language models perform on sentiment classification tasks in comparison to other techniques.

By transforming raw text-based data into actionable insights, this project enables the comprehension of customers' general sentiments, detects prevalent problems (like application crashes and lag), and emphasizes positive traits (such as usability and application layout). The truths obtained from these observations help application developers allocate priority rectifications and improvements, as well as empower business leaders with the proper facts for making sound decisions directed at enhancing user retention and building brand loyalty.

2. Project Overview

This project, titled "**Sentiment Analysis of Uber Customer Reviews Using Machine Learning and Deep Learning Techniques**", focuses on analyzing customer feedback to uncover sentiment patterns and generate meaningful insights. The main objective is to classify user reviews into three sentiment categories: positive, neutral, and negative. The classification is achieved by applying a combination of traditional machine learning algorithms, deep learning architectures, and transformer-based models.

The dataset used contains over 12,000 anonymized Uber reviews collected from the Google Play Store. The project involves several phases including data preprocessing, feature engineering, model training, and performance evaluation. Three traditional models are employed in this study: Logistic Regression, Support Vector Machine, and Random Forest. For deep learning, the BiLSTM model is implemented, while the transformer-based model DistilBERT is used to capture contextual language representations.

The goal is to evaluate and compare the effectiveness of these models in accurately predicting sentiments. This demonstrates the application of NLP techniques in handling large volumes of unstructured textual data. The findings of this project can assist developers in identifying common user concerns, improving app functionality, and guiding businesses in making informed decisions to enhance user satisfaction and loyalty.

3. Problem Statement

The growth of reliance on ride-hailing apps, like Uber, has made customer reviews an important aspect in improving service quality. Places like the Google Play Store collect huge amounts of user feedback that capture a range of sentiments, both praises and complaints. However, the manual evaluation of this enormous amount of unstructured textual data is tedious, inefficient, and infeasible for large-scale applications.

Traditional data analysis tools are not sufficient to fully grasp the emotional subtleties and context complexities inherent in natural language. For this reason, there is a critical need for automatic sentiment analysis techniques that can classify customer opinions into meaningful categories, the positive, neutral, or negative. These classifications help service providers identify performance deficiency areas, understand the level of user satisfaction, and enhance the overall effectiveness of applications.

This study investigates the problem from the perspective of NLP and machine learning methods for sentiment classification in reviews related to Uber. The goal is to analyze the performance of traditional machine learning (ML) models against state-of-the-art deep learning and transformer-based models in order to determine the best method for sentiment analysis in large datasets.

4. Literature Review

In this era, sentiment analysis as a powerful tool to evaluate user satisfaction, app usability, and service reliability in ride-hailing platforms (e.g. Uber, Grab, Lyft) [1] Sentiment analysis, or opinion mining, is a subdiscipline of the study of NLP that examines the emotional tone contained in textual data, typically labeling user feedback as positive, negative, or neutral. Through the conversion of raw application store ratings to structured sentiment categories, companies can systematically pull-out insight from large volumes of feedback that would be impractical to review through human judgment. For instance, the Google Play Store gathers multiple reviews for ride-hailing apps, representing real user experience ranging from technical issues (like app crashes and slow performance) to complaints about driver behavior and fares [1].

The use of automated sentiment analysis has become prominent as a method to systematically analyze reviews and enable data-driven improvements. Over the last few years, a range of ML DL methods have been used by researchers to analyze feedback on ride-hailing apps. Traditional classification methods, such as Logistic Regression, SVM, and Random Forests, have been used for the task of review sentiment prediction, along with more advanced architectures like recurrent neural networks, namely *BiLSTM*, and Transformer-based models like BERT and its distilled version, DistilBERT.

Having outlined the overall trends, next examine representative academic works that employed the specific classification techniques used in this project: Logistic Regression, SVM, Random Forest, BiLSTM, and DistilBERT. Each of the sections below presents how these models were applied to ride-hailing review data and the main findings documented.

Logistic Regression

Logistic regression is often used as a base classifier in sentiment analysis in reviews for applications. [1] used logistic regression among several other models in a comparison study of 12,052 reviews that were collected from the Google Play Store relating to Uber, Grab, and other ride-hailing apps. In the study mentioned above, logistic regression showed respectable accuracy, in the mid-70% range, when used to classify review sentiment, but was slightly outperformed by some other classifiers, notably support vector machines (SVM). Similarly, [2] presented an analytic comparison of the performance of different machine learning algorithms, among them logistic regression, on ride-hailing service reviews. They highlighted that logistic regression sets a strong baseline, yet it can be improved by using more complex models, as further discussed in their results.

Support Vector Machine (SVM)

SVM have now become the method of choice for performing sentiment analysis of customer reviews owing to their remarkable efficiency in processing textual data. In the above-stated study by [3], SVM was the top-performing conventional machine learning algorithm used to analyze sentiments of ride-share app reviews with an accuracy rate of nearly 76.7% on a multi-platform dataset of reviews. The performance registered herein was the peak in comparison with conventional models like Naive Bayes, logistic regression, and Random Forest on the same dataset. The effectiveness of SVM has been reported across several studies; for instance, [3] carried out Twitter sentiment analysis of Uber and Ola using SVM combined with Naïve Bayes, highlighting the greater accuracy of SVM in classifying tweets about ride-hailing companies. Overall, studies indicate SVM often outperforms other standalone ML classifiers for textual sentiment tasks in this domain, especially when combined with proper feature extraction (e.g. TF-IDF or n-grams).

Random Forest

Ensemble methods like Random Forest have also been used in sentiment analysis of ride-hailing reviews, often with good results. For example, [1] carried out a study of reviews of the Grab ride-hailing app using three different classifiers (Random Forest, SVM, and Naive Bayes). Their experiments showed that Random Forest achieved the highest accuracy (around 95.14%) in classifying Google Play Store reviews of the Grab app [4]. This suggests that the ensemble structure of decision trees can successfully recognize subtle patterns in customer reviews. However, the cross-platform study by [1] showed that Random Forest was marginally less accurate compared to SVM when run on a larger dataset, [2] though it was still among the better performers. The high accuracy reported in the Grab-specific study might be due to dataset characteristics or feature engineering steps, but it underscores that Random Forest can be a very effective model for sentiment analysis in ride-hailing contexts when tuned properly

Bidirectional LSTM (BiLSTM)

Recurrent neural networks, in particular the Long Short-Term Memory (LSTM) architecture and its bidirectional variation, have received considerable attention in recent years for performing sentiment analysis on text reviews. The BiLSTM has the special ability to discern context from both the previous and following words in a sentence, which is particularly important in understanding sentiment in complicated texts, such as app feedback. [5] created a sentiment classification system specifically for Uber and Ola reviews by utilizing a single-layer Bidirectional LSTM model, illustrating that this system was both computationally efficient and effective in enabling real-time sentiment analysis. The chosen model is based on the BiLSTM structure, and it was ascertained that it is effective and capable in representing sequential contexts. In a similar study, [1] compared the performance of BiLSTM with LSTM, GRU, and several other approaches using thousands of reviews on ride-share apps; while their top-performing model was a BiGRU, the inclusion of BiLSTM in their study highlighted the strength of BiLSTM as a powerful deep-learning method for sentiment analysis tasks.

Transformer Models (DistilBERT/BERT)

Transformer-based models represent the state-of-the-art technology in text sentiment analysis, e.g., ratings of ride-hailing apps. BERT and its multi-head self-attention yield very rich context-sensitive embeddings with a dramatic boost in sentiment classification performance. A smaller version, DistilBERT, has been particularly helpful when computational efficiency is wanted without giving up too much accuracy. Within the ride-hailing scenario, [1] trained a DistilBERT model with 1,818 Uber/Grab reviews and attained a very high 98.84% classification accuracy that significantly outperformed their CNN and LSTM models.

This nearly perfect accuracy indicates the successful ability of transformer models to capture the fine-grained language used in user comments (with quite modest training sets). In the same vein, found that a classifier based on BERT had the highest precision in their experiments in sentiment detection (performing better than several conventional algorithms) [6].

The benefit of transformers is that they can learn to understand context at a deep level e.g., that "no lag now" in a review is positive, while "no improvement" is negative, from word interaction. They leverage knowledge from pre-training on massive text corpora, which is then fine-tuned on domain-specific data (like app reviews). The result is a model that not only learns domain terminology quickly (e.g. “surge pricing”, “driver cancel”) but also grasps semantic subtleties and sentiment cues that simpler models might miss [6]. The literature now increasingly reports BERT and its variants as the top-performing models for sentiment analysis across many domain [7]. The only limitation are that these models are resource-intensive and are complex. Yet, because sentiment analysis provides useful information to ride-hailing companies, applying transformers is generally worth the hassle. In brief, transformer models such as BERT/DistilBERT are now the best method to label ride-hailing app reviews' emotions. They enable developers and researchers to obtain very accurate information about users' satisfaction and issues from a tremendous amount of review data.

5. Methodology

This project employs a straightforward manner of converting raw app review data into meaningful feelings insights. It achieves this by employing NLP, simple machine learning, deep learning, and transformer models. It breaks down the entire process into these broad steps.

Data Collection

The study examines over 12,000 anonymous customer app reviews of Uber on the Google Play Store. The reviews contain written comments, ratings of 1 to 5 stars, developer replies, and dates. The data was structured and verified to maintain its ethical nature and to safeguard its quality.

Data Preprocessing

Data cleaning steps were applied to remove null values, special characters, emojis, and non-English content. Standard NLP preprocessing techniques were employed, including:

- Tokenization
- Lowercasing
- Removal of stopwords (using NLTK)
- Lemmatization (using WordNetLemmatizer)
- Punctuation and number filtering

The preprocessed text was then converted into numerical representations using:

- TF-IDF vectorization for traditional ML models
- Word embedding (e.g., Tokenizer + padded sequences) for BiLSTM
- Tokenized input with attention masks for DistilBERT

Sentiment Labeling

Each review was assigned a sentiment label based on its numeric rating:

- **Positive**: Ratings of 4 and 5
- **Neutral**: Rating of 3
- **Negative**: Ratings of 1 and 2

These labels were used as the target variable for supervised model training.

Model Training

The following models were trained and evaluated:

- **Traditional Machine Learning Models**:
  - Logistic Regression
  - SVM
  - Random Forest (These models used the TF-IDF features for classification and were evaluated via k-fold cross-validation where k=5.)

- **Deep Learning Model**:
    - BiLSTM (Implemented using Keras and TensorFlow, this model was trained with word-embedded sequences and evaluated using categorical cross-entropy loss and accuracy metrics.)
- **Transformer Model**:
    - DistilBERT (Fine-tuned using HuggingFace Transformers, DistilBERT was trained on the review text and evaluated using a stratified split and classification metrics.)

Evaluation Metrics

Each model was evaluated using the following metrics to assess performance:

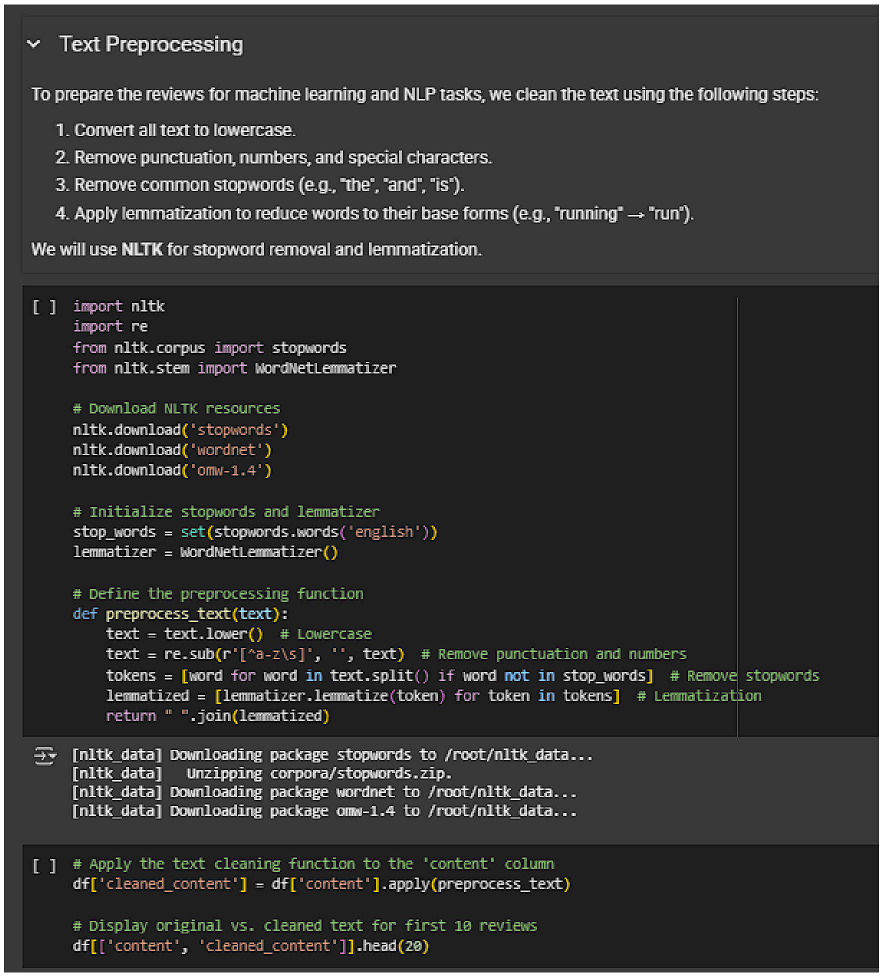
- Accuracy
- Precision
- Recall
- F1-Score

6. Sentiment Analysis

6.1 Text Preprocessing

Sentiment analysis, also known as opinion mining, plays a central role in this project by transforming unstructured customer reviews from the Uber app into structured sentiment categories: positive, neutral, and negative. The goal is to understand users' emotional responses and derive insights that can improve service quality and user satisfaction.

To ensure accurate analysis, the dataset was first cleaned. Reviews with fewer than three words were removed to eliminate noise and low-information content. The remaining text was converted to lowercase and stripped of special characters. Stopwords were removed, and stemming or lemmatization was applied where relevant. Figure 6.1 Showing the text preprocessing code and the result.



Text Preprocessing

To prepare the reviews for machine learning and NLP tasks, we clean the text using the following steps:

1. Convert all text to lowercase.
2. Remove punctuation, numbers, and special characters.
3. Remove common stopwords (e.g., "the", "and", "is").
4. Apply lemmatization to reduce words to their base forms (e.g., "running" → "run").

We will use **NLTK** for stopword removal and lemmatization.

```
[ ] import nltk
    import re
    from nltk.corpus import stopwords
    from nltk.stem import WordNetLemmatizer

    # Download NLTK resources
    nltk.download('stopwords')
    nltk.download('wordnet')
    nltk.download('omw-1.4')

    # Initialize stopwords and lemmatizer
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    # Define the preprocessing function
    def preprocess_text(text):
        text = text.lower()  # Lowercase
        text = re.sub(r'[^a-z\s]', '', text)  # Remove punctuation and numbers
        tokens = [word for word in text.split() if word not in stop_words]  # Remove stopwords
        lemmatized = [lemmatizer.lemmatize(token) for token in tokens]  # Lemmatization
        return " ".join(lemmatized)
```

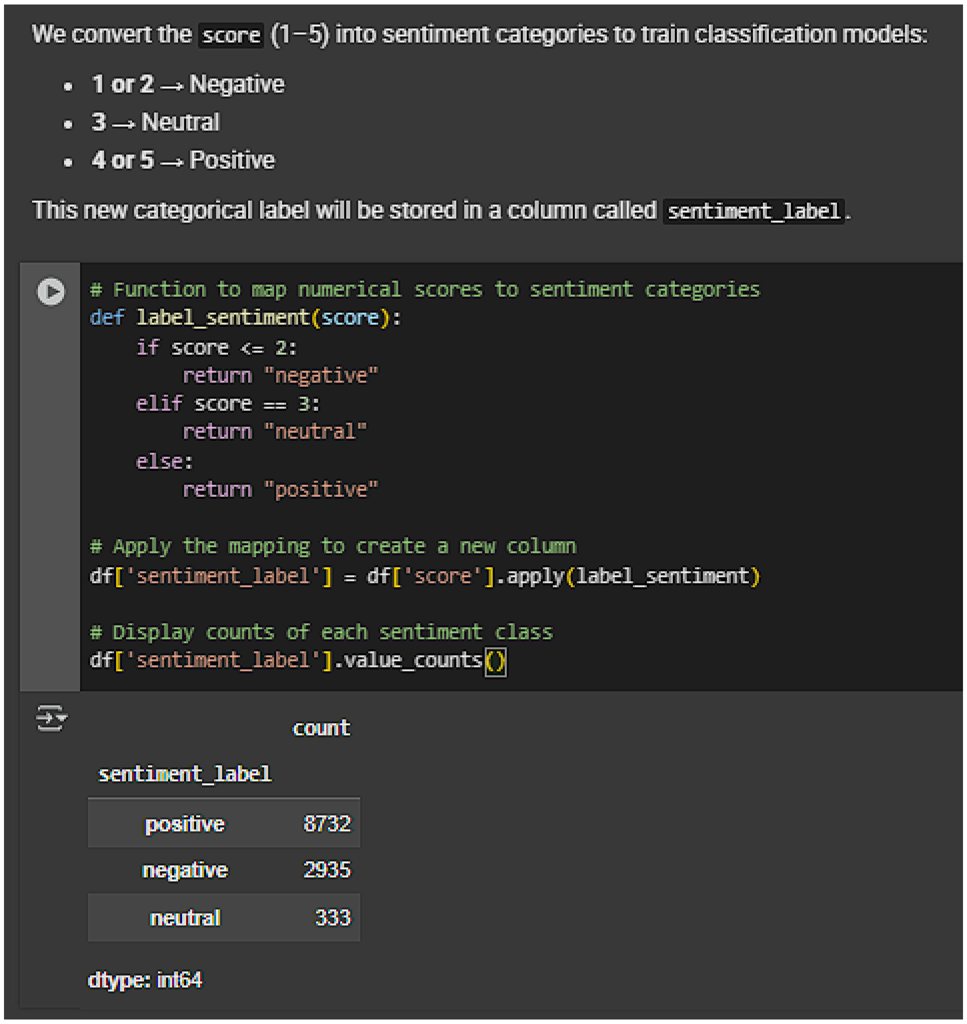
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

```
[ ] # Apply the text cleaning function to the 'content' column
    df['cleaned_content'] = df['content'].apply(preprocess_text)

    # Display original vs. cleaned text for first 10 reviews
    df[['content', 'cleaned_content']].head(20)
```

Figure 6.1 Text Preprocessing

Next, to complete the sentiment analysis, each review in the dataset includes a numerical score from **1 to 5**, which represent the user's level of satisfaction. To enable **supervised learning**, these numeric ratings were converted into **categorical sentiment labels** using a custom function. Figure 6.2 illustrated the logic was as follows:

- Ratings **1 or 2** were classified as **negative** sentiment.
- Rating **3** was considered **neutral**.
- Ratings **4 or 5** were classified as **positive** sentiment.


We convert the `score` (1–5) into sentiment categories to train classification models:

- **1 or 2** → Negative
- **3** → Neutral
- **4 or 5** → Positive

This new categorical label will be stored in a column called `sentiment_label`.

```
# Function to map numerical scores to sentiment categories
def label_sentiment(score):
    if score <= 2:
        return "negative"
    elif score == 3:
        return "neutral"
    else:
        return "positive"

# Apply the mapping to create a new column
df['sentiment_label'] = df['score'].apply(label_sentiment)

# Display counts of each sentiment class
df['sentiment_label'].value_counts()
```

```
                 count
sentiment_label
   positive       8732
   negative       2935
   neutral         333

dtype: int64
```

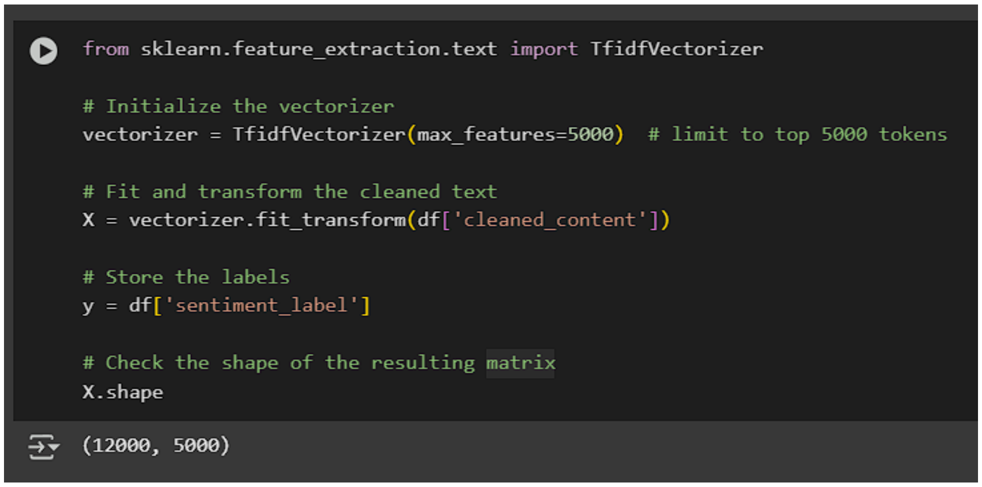
Figure 6.2 Sentiment Labal

Based on the figure 6.2 above, the majority of users left good feedback, but others were neither good nor bad. It was addressed during model training using techniques such as class weighting to prevent biased predictions.

6.2 Feature Engineering

TF-IDF Vectorization

For all traditional models such as Logistic Regression, SVM, and Random Forest, we converted the review text to TF-IDF vectors. This method determines how significant words in a review are in relation to all reviews. This reduces the influence of typical words that are generally not useful (such as "the" and "and"). The tiny TF-IDF score matrix was used as input to these typical classifiers. Figure 6.3 Show the TF-IDF implementation on this project.



```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the vectorizer
vectorizer = TfidfVectorizer(max_features=5000)  # limit to top 5000 tokens

# Fit and transform the cleaned text
X = vectorizer.fit_transform(df['cleaned_content'])

# Store the labels
y = df['sentiment_label']

# Check the shape of the resulting matrix
X.shape
```

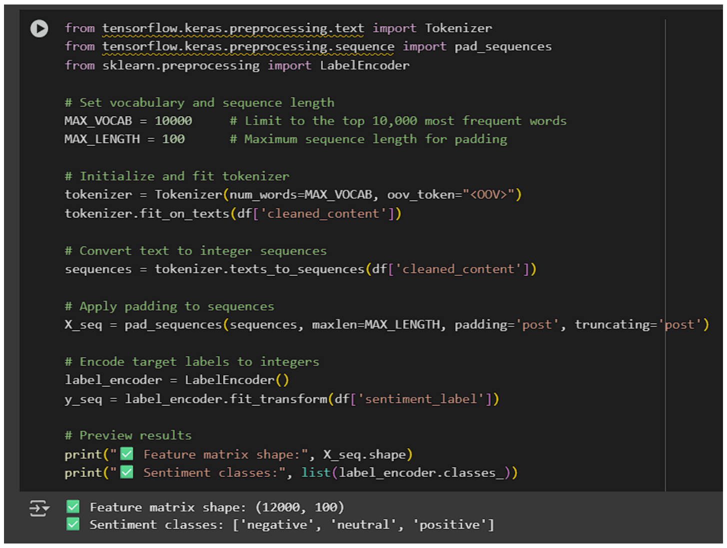
```
(12000, 5000)
```

Figure 6.3 TF-IDF

Token Sequences (for BiLSTM Model)

The BiLSTM model requires a list of numbers for every word in the reviews. Figure 6.4 Shows the Tokenizer implementation. In order to prepare the data:

- Keras Tokenizer splits the
  text into items known as tokens and assigns a number to every unique word.
- All the sequences were created of equal length so that their sizes were the same. This allow the model to process in more efficient



```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder

# Set vocabulary and sequence length
MAX_VOCAB = 10000      # Limit to the top 10,000 most frequent words
MAX_LENGTH = 100       # Maximum sequence length for padding

# Initialize and fit tokenizer
tokenizer = Tokenizer(num_words=MAX_VOCAB, oov_token="<OOV>")
tokenizer.fit_on_texts(df['cleaned_content'])

# Convert text to integer sequences
sequences = tokenizer.texts_to_sequences(df['cleaned_content'])

# Apply padding to sequences
X_seq = pad_sequences(sequences, maxlen=MAX_LENGTH, padding='post', truncating='post')

# Encode target labels to integers
label_encoder = LabelEncoder()
y_seq = label_encoder.fit_transform(df['sentiment_label'])

# Preview results
print("✅ Feature matrix shape:", X_seq.shape)
print("✅ Sentiment classes:", list(label_encoder.classes_))
```

```
✅ Feature matrix shape: (12000, 100)
✅ Sentiment classes: ['negative', 'neutral', 'positive']
```

Figure 6.4 Tokenizer

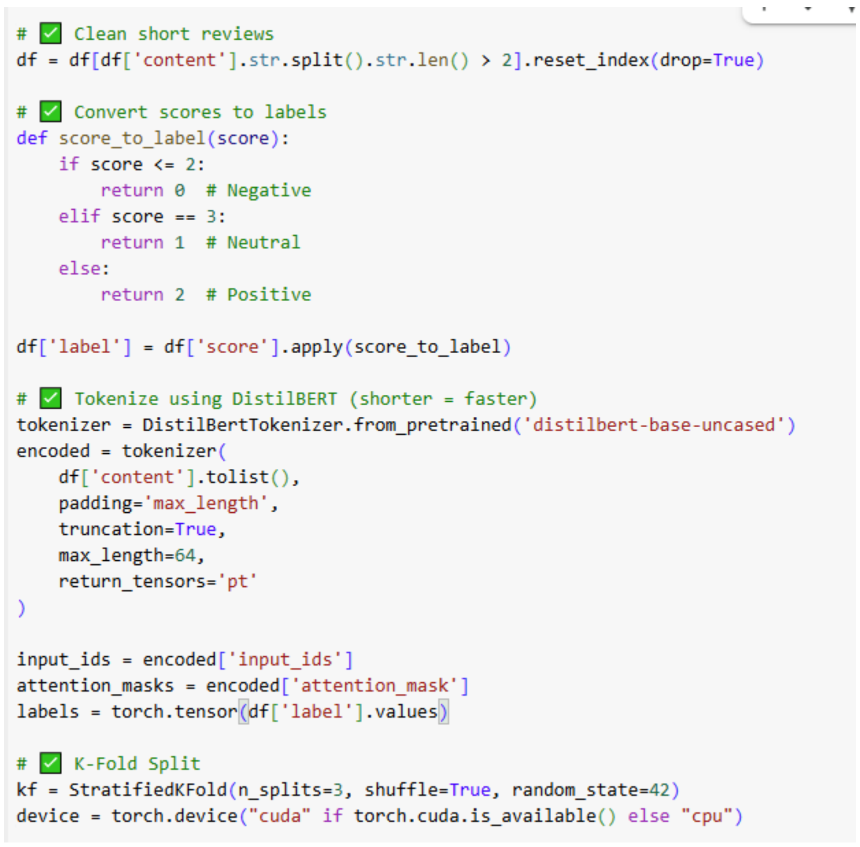
Transformers Input Format (for DistilBERT)

To train the DistilBERT transformer model, we took the setup steps depicted in Figure 6.5, where input reviews needed to be preprocessed for transformer-based models: The tokenization and preparation followed the implementation pipeline proposed in HuggingFace to make the best use of the pretrained model. In order to eliminate unqualified inputs and to guarantee the analysis being productive, extremely short reviews of less than three words were removed. After that, each review's numeric score (between 1 and 5) was transformed into categorical sentiment labels with 1 and 2 labeled as Negative (0) or 3 as Neutral (1) or 4 and 5 as Positive (2). Tokenization of the review texts was performed by using the DistilBertTokenizer from the Transformers library by HuggingFace, which breaks down the input sentences into subword units (WordPieces) so that all input tokens matched the DistilBERT vocabulary.

The following were given as inputs to each of the reviews:

- Input IDs: The numerical representation of tokenized words;
- Attention Masks: Binary sequences whereby tokens containing actual content are marked as a 1, whereas padding tokens are assigned 0 so that these can be ignored by the model during attention computations.

For making efficient training paramount, sequence length was capped at 64 tokens. Afterward, the obtained input_ids and attention_mask were used as inputs for further DistilBERT classification, and the sentiment labels were used as ground truths in classification. Subsequently, a 3-fold Stratified K-Fold Cross-Validation split was performed on the dataset to maintain the original class proportions in negative, neutral, and positive sentiment classes across each fold. This step empowered the assessment of the model's capability to generalize and its stability on various data subsets.

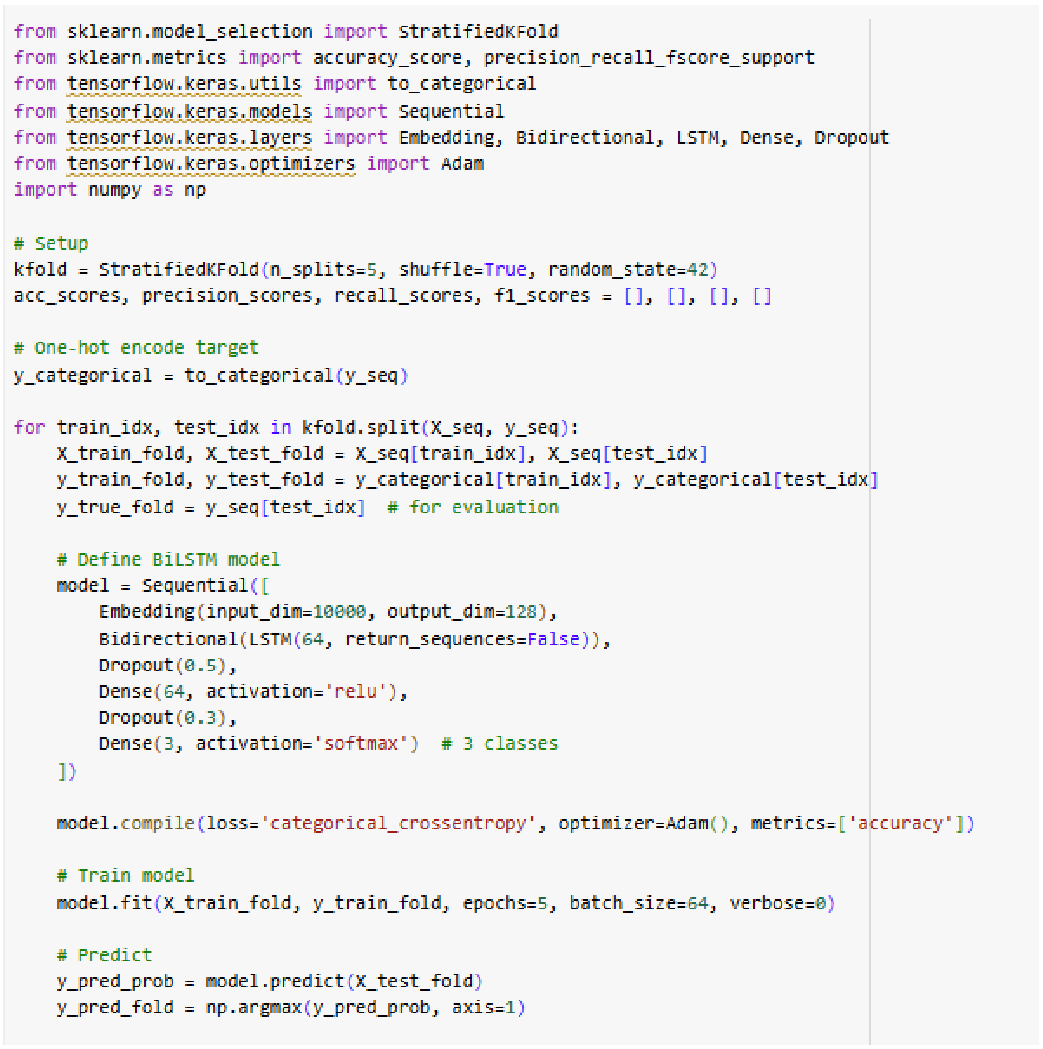


```
# ✅ Clean short reviews
df = df[df['content'].str.split().str.len() > 2].reset_index(drop=True)

# ✅ Convert scores to labels
def score_to_label(score):
    if score <= 2:
        return 0  # Negative
    elif score == 3:
        return 1  # Neutral
    else:
        return 2  # Positive

df['label'] = df['score'].apply(score_to_label)

# ✅ Tokenize using DistilBERT (shorter = faster)
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
encoded = tokenizer(
    df['content'].tolist(),
    padding='max_length',
    truncation=True,
    max_length=64,
    return_tensors='pt'
)

input_ids = encoded['input_ids']
attention_masks = encoded['attention_mask']
labels = torch.tensor(df['label'].values)

# ✅ K-Fold Split
kf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Figure 6.5 Transformers Input

7. Model Training

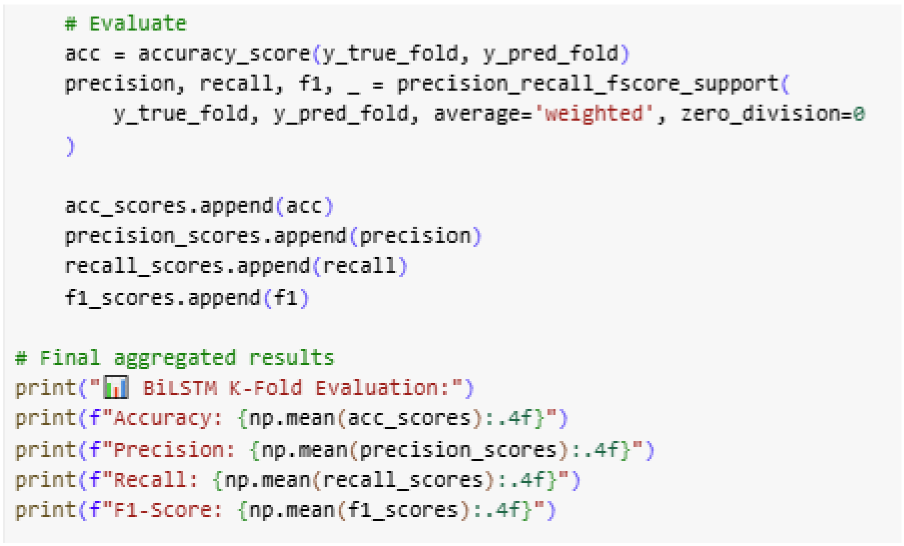
Train Deep Learning Models: BiLSTM



```python
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import numpy as np

# Setup
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
acc_scores, precision_scores, recall_scores, f1_scores = [], [], [], []

# One-hot encode target
y_categorical = to_categorical(y_seq)

for train_idx, test_idx in kfold.split(X_seq, y_seq):
    X_train_fold, X_test_fold = X_seq[train_idx], X_seq[test_idx]
    y_train_fold, y_test_fold = y_categorical[train_idx], y_categorical[test_idx]
    y_true_fold = y_seq[test_idx]  # for evaluation

    # Define BiLSTM model
    model = Sequential([
        Embedding(input_dim=10000, output_dim=128),
        Bidirectional(LSTM(64, return_sequences=False)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(3, activation='softmax')  # 3 classes
    ])

    model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

    # Train model
    model.fit(X_train_fold, y_train_fold, epochs=5, batch_size=64, verbose=0)

    # Predict
    y_pred_prob = model.predict(X_test_fold)
    y_pred_fold = np.argmax(y_pred_prob, axis=1)
```

Figure 6.6 BiLSTM training

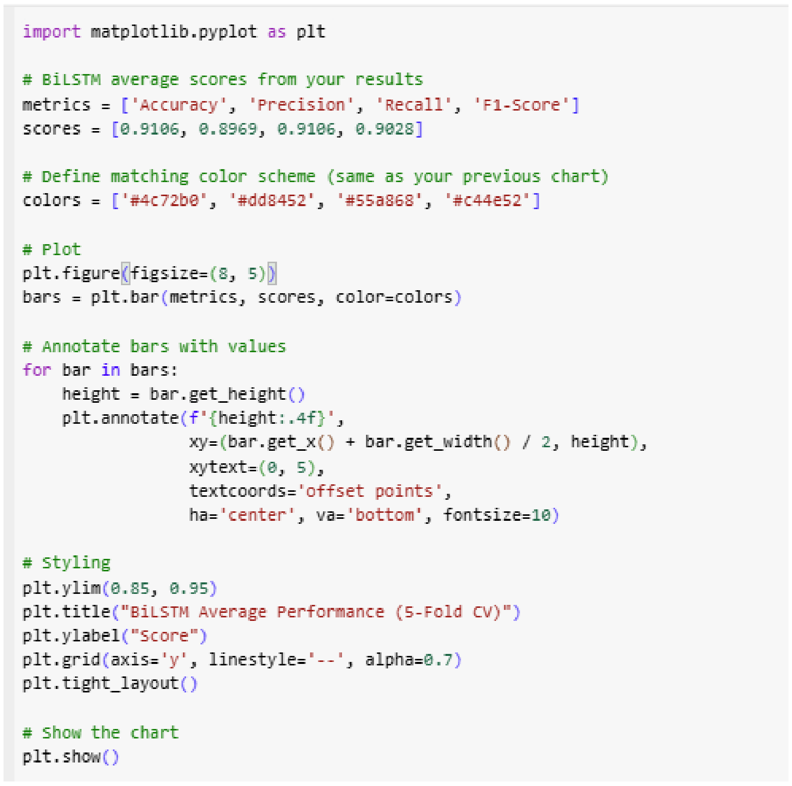
The code shown in Figure 6.6 runs 5-fold stratified cross-validation for the assessment of a Bidirectional LSTM model for multi-class text classification. The target label (y_seq) is one-hot encoded for usage with soft-max output layers. Basically, folds of training and testing splits are kept balanced for class distribution using StratifiedKFold. The BiLSTM model consists of an embedding layer, a bidirectional LSTM, dropout to regularize, and dense layers to a 3-class softmax output. It compiles the model using categorical cross-entropy loss and then fits silently (verbose=0) for 5 epochs.



```python
    # Evaluate
    acc = accuracy_score(y_true_fold, y_pred_fold)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y_true_fold, y_pred_fold, average='weighted', zero_division=0
    )

    acc_scores.append(acc)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)

# Final aggregated results
print("📊 BiLSTM K-Fold Evaluation:")
print(f"Accuracy: {np.mean(acc_scores):.4f}")
print(f"Precision: {np.mean(precision_scores):.4f}")
print(f"Recall: {np.mean(recall_scores):.4f}")
print(f"F1-Score: {np.mean(f1_scores):.4f}")
```

Figure 6.7 BiLSTM Evaluation

After training, predictions are made on the validation set, and metrics which are accuracy, precision, recall, and F1-score. These metrics are computed and stored for each fold. Finally, the average of these metrics is printed to summarize the overall model performance across all 5 folds.

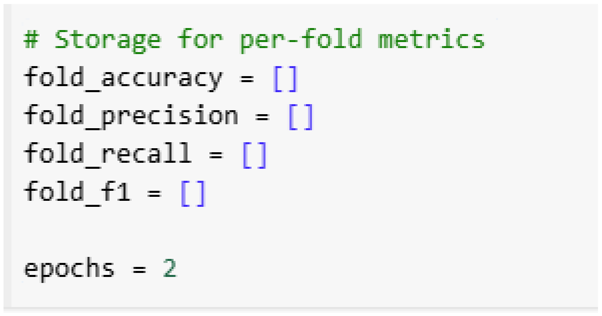


```python
import matplotlib.pyplot as plt

# BiLSTM average scores from your results
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
scores = [0.9106, 0.8969, 0.9106, 0.9028]

# Define matching color scheme (same as your previous chart)
colors = ['#4c72b0', '#dd8452', '#55a868', '#c44e52']

# Plot
plt.figure(figsize=(8, 5))
bars = plt.bar(metrics, scores, color=colors)

# Annotate bars with values
for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height:.4f}',
                 xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 5),
                 textcoords='offset points',
                 ha='center', va='bottom', fontsize=10)

# Styling
plt.ylim(0.85, 0.95)
plt.title("BiLSTM Average Performance (5-Fold CV)")
plt.ylabel("Score")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Show the chart
plt.show()
```

Figure 6.8 BiLSTM Visualization

The code above in figure 6.8 plots a bar chart to visualize the average performance metrics of BiLSTM models when evaluated with 5-fold cross-validation. Performance metrics include Accuracy, Precision, Recall, and F1-Score, with average values attached for each. Each bar takes colors from the standard palette used for other plots in earlier sections, while numeric scores are given over each bar for clarity. The y-axis is kept limited in the range 0.85–0.95 to highlight the performance differences, along with grid lines to aid readability. This plot intuitively summarizes the performance of the BiLSTM model across metrics weldedly.

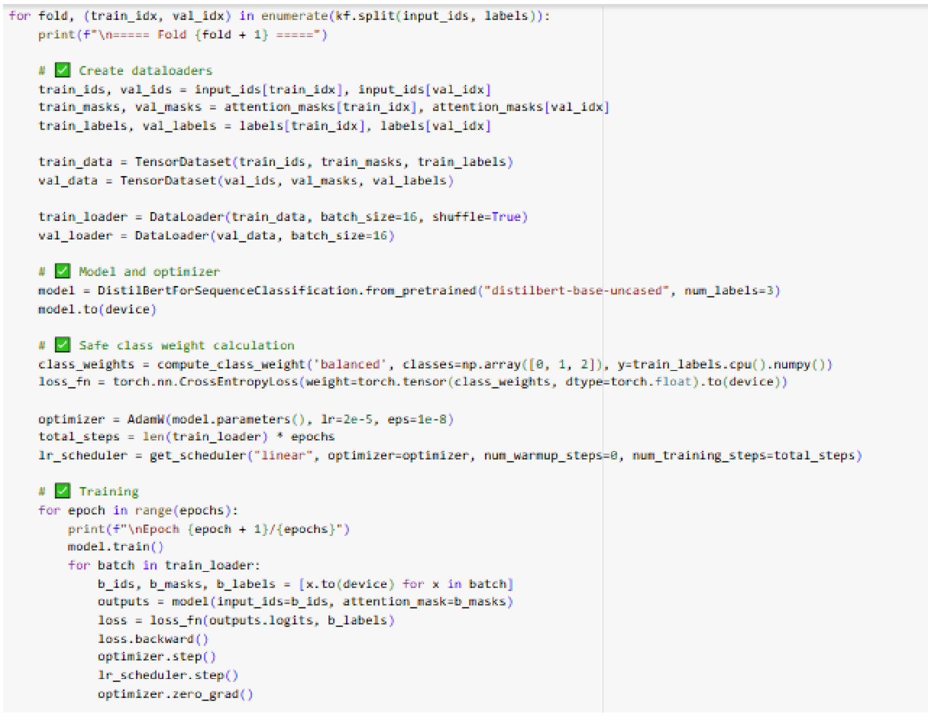
Transfomer Model (DistilBert)



```
# Storage for per-fold metrics
fold_accuracy = []
fold_precision = []
fold_recall = []
fold_f1 = []

epochs = 2
```

Figure 6.9 Metrics' Storage

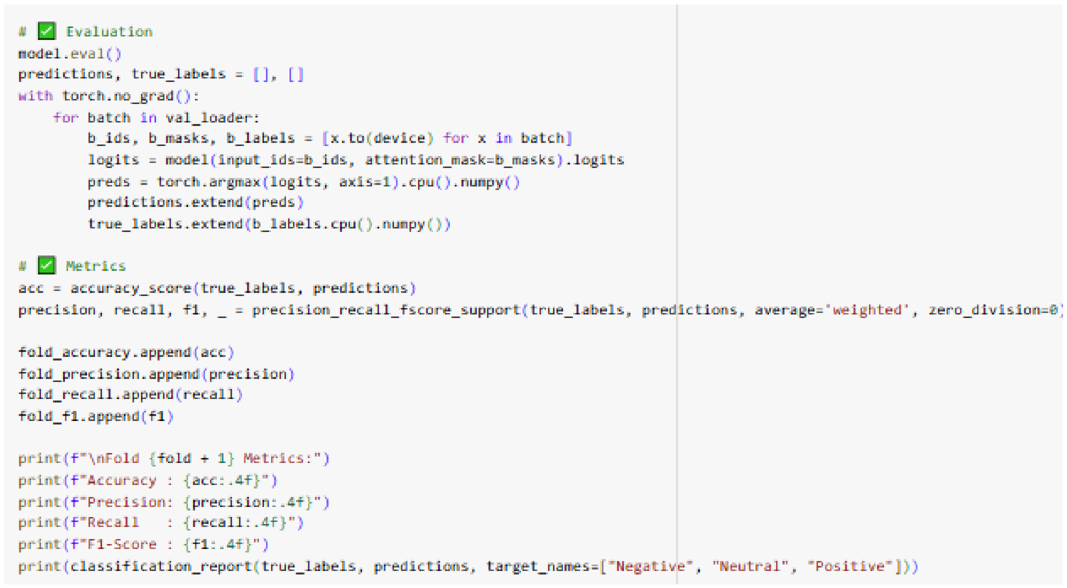
These lists will store the evaluation metrics (Accuracy, Precision, Recall, F1-score) for each fold of cross-validation so that you can compute the average performance across all folds after training.



```python
for fold, (train_idx, val_idx) in enumerate(kf.split(input_ids, labels)):
    print(f"\n===== Fold {fold + 1} =====")

    # ✅ Create dataloaders
    train_ids, val_ids = input_ids[train_idx], input_ids[val_idx]
    train_masks, val_masks = attention_masks[train_idx], attention_masks[val_idx]
    train_labels, val_labels = labels[train_idx], labels[val_idx]

    train_data = TensorDataset(train_ids, train_masks, train_labels)
    val_data = TensorDataset(val_ids, val_masks, val_labels)

    train_loader = DataLoader(train_data, batch_size=16, shuffle=True)
    val_loader = DataLoader(val_data, batch_size=16)

    # ✅ Model and optimizer
    model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=3)
    model.to(device)

    # ✅ Safe class weight calculation
    class_weights = compute_class_weight('balanced', classes=np.array([0, 1, 2]), y=train_labels.cpu().numpy())
    loss_fn = torch.nn.CrossEntropyLoss(weight=torch.tensor(class_weights, dtype=torch.float).to(device))

    optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
    total_steps = len(train_loader) * epochs
    lr_scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=total_steps)

    # ✅ Training
    for epoch in range(epochs):
        print(f"\nEpoch {epoch + 1}/{epochs}")
        model.train()
        for batch in train_loader:
            b_ids, b_masks, b_labels = [x.to(device) for x in batch]
            outputs = model(input_ids=b_ids, attention_mask=b_masks)
            loss = loss_fn(outputs.logits, b_labels)
            loss.backward()
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
```

Figure 7.0 DistilBERT training

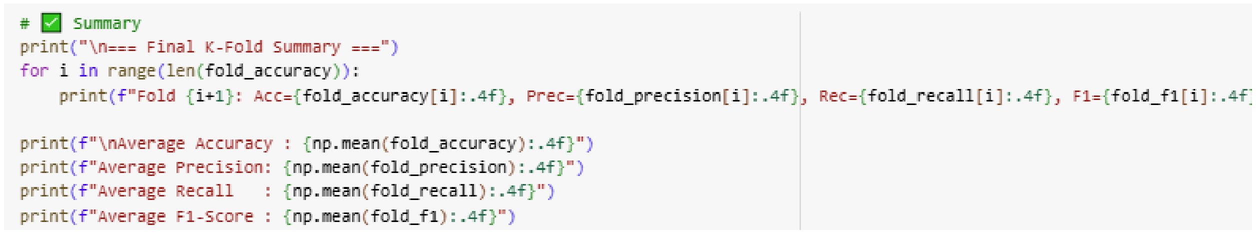
This code implements **3-fold cross-validation** to evaluate the performance of a DistilBERT-based sentiment classifier. For each fold, it splits the data into training and validation sets while preserving the class distribution using `StratifiedKFold`. The input reviews are tokenized into `input_ids` and `attention_masks`, and paired with their sentiment labels to create PyTorch `TensorDataset`s and `DataLoader`s for efficient batching. A new DistilBERT model is loaded for each fold and fine-tuned for 2 epochs using a weighted cross-entropy loss function to handle class imbalance. The model is trained with the AdamW optimizer and a linear learning rate scheduler.



```
# ✅ Evaluation
model.eval()
predictions, true_labels = [], []
with torch.no_grad():
    for batch in val_loader:
        b_ids, b_masks, b_labels = [x.to(device) for x in batch]
        logits = model(input_ids=b_ids, attention_mask=b_masks).logits
        preds = torch.argmax(logits, axis=1).cpu().numpy()
        predictions.extend(preds)
        true_labels.extend(b_labels.cpu().numpy())

# ✅ Metrics
acc = accuracy_score(true_labels, predictions)
precision, recall, f1, _ = precision_recall_fscore_support(true_labels, predictions, average='weighted', zero_division=0

fold_accuracy.append(acc)
fold_precision.append(precision)
fold_recall.append(recall)
fold_f1.append(f1)

print(f"\nFold {fold + 1} Metrics:")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print(f"F1-Score : {f1:.4f}")
print(classification_report(true_labels, predictions, target_names=["Negative", "Neutral", "Positive"]))
```

Figure 7.1 DistilBERT Evaluation

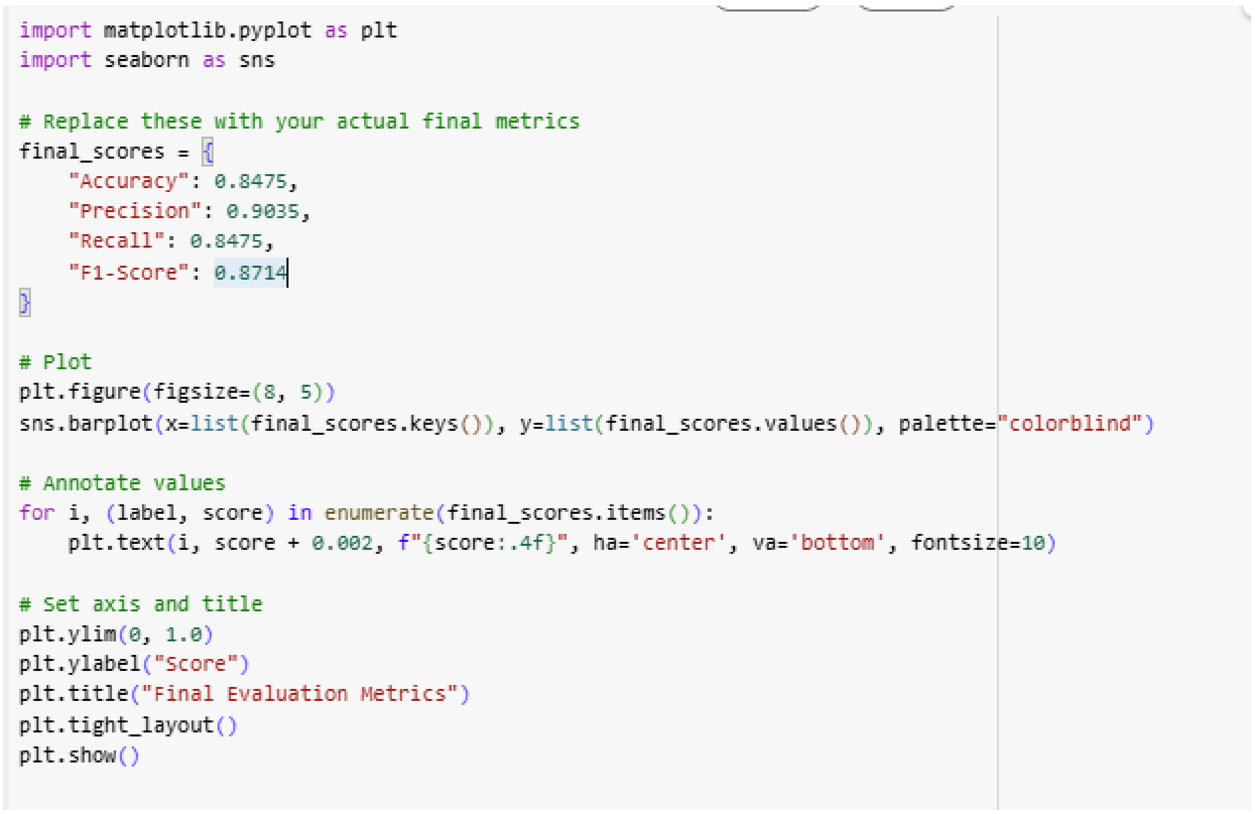
After training, it is evaluated on the validation set, and predictions are compared to the true labels using accuracy, precision, recall, and F1-score. These metrics are stored per fold and printed along with a detailed classification report for all three sentiment classes which are “Negative”,” Neutral”, and “Positive”.



```
# ✅ Summary
print("\n=== Final K-Fold Summary ===")
for i in range(len(fold_accuracy)):
    print(f"Fold {i+1}: Acc={fold_accuracy[i]:.4f}, Prec={fold_precision[i]:.4f}, Rec={fold_recall[i]:.4f}, F1={fold_f1[i]:.4f

print(f"\nAverage Accuracy : {np.mean(fold_accuracy):.4f}")
print(f"Average Precision: {np.mean(fold_precision):.4f}")
print(f"Average Recall   : {np.mean(fold_recall):.4f}")
print(f"Average F1-Score : {np.mean(fold_f1):.4f}")
```

Figure 7.2 K-fold cross-validation DistilBERT

This code prints a detailed summary of the performance metrics for each fold of the K-Fold Cross-Validation conducted on DistilBERT. For every fold, it displays the accuracy, precision, recall, and F1-Score, each rounded to four decimal places. After listing per-fold results, it computes and prints the average of each metric across all folds, providing an overall assessment of the model’s consistency and generalization ability. This summary helps evaluate whether the model performs reliably across different data splits.



```python
import matplotlib.pyplot as plt
import seaborn as sns

# Replace these with your actual final metrics
final_scores = {
    "Accuracy": 0.8475,
    "Precision": 0.9035,
    "Recall": 0.8475,
    "F1-Score": 0.8714
}

# Plot
plt.figure(figsize=(8, 5))
sns.barplot(x=list(final_scores.keys()), y=list(final_scores.values()), palette="colorblind")

# Annotate values
for i, (label, score) in enumerate(final_scores.items()):
    plt.text(i, score + 0.002, f"{score:.4f}", ha='center', va='bottom', fontsize=10)

# Set axis and title
plt.ylim(0, 1.0)
plt.ylabel("Score")
plt.title("Final Evaluation Metrics")
plt.tight_layout()
plt.show()
```

Figure 7.3 DistilBERT Evaluation

This code creates a bar chart to visualize the final evaluation metrics which are accuracy, precision, recall, and F1-score of DistilBERT. It uses the seaborn library to plot the bars with a colorblind-friendly palette for accessibility. The y-axis is scaled from 0 to 1.0 to represent metric scores as proportions, and each bar is annotated with its exact value for clarity. This visual summary provides an at-a-glance understanding of how well the model performed across multiple key performance indicators, making it easier to interpret and compare results.

8. Results and Visualization

This chapter outlines the main results derived from sentiment analysis carried out on Uber reviews, including model performance comparisons, information about sentiment distribution, and graphical representation of common keywords and bigrams present in user comments.

8.1 Sentiment Distribution

Uber sentiment analysis captures user satisfaction in the proportion of positive, negative, or neutral feedback, helping the company assess service performance and sentiment. Figure 8.1 presents the overall sentiment distribution of the collected Uber reviews. The largest chunk of these reviews was labeled positive (8,732), followed by negative labels (2,935), while a very small portion was labeled as neutral (333). This shows that a considerable percentage of users are likely to express a positive experience with the app.

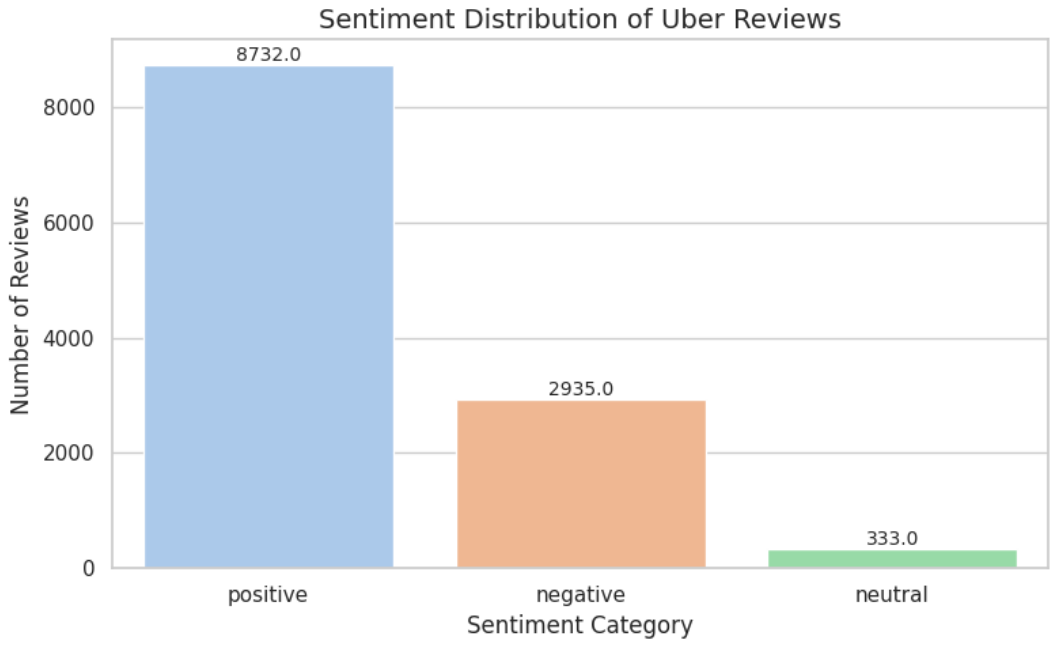


Figure 8.1 Sentiment Distribution

8.2 Model Performance Comparison

A comparative analysis of five sentiment classification models was conducted: Logistic Regression, SVM, Random Forest, BiLSTM, and DistilBERT. Evaluation was based on Accuracy, Precision, Recall, and F1-Score for the fair and valid comparison. Figure 8.2 Showing the result in the bar chart and Table 8.1 the results are visualized using a bar chart to highlight the performance of each model.

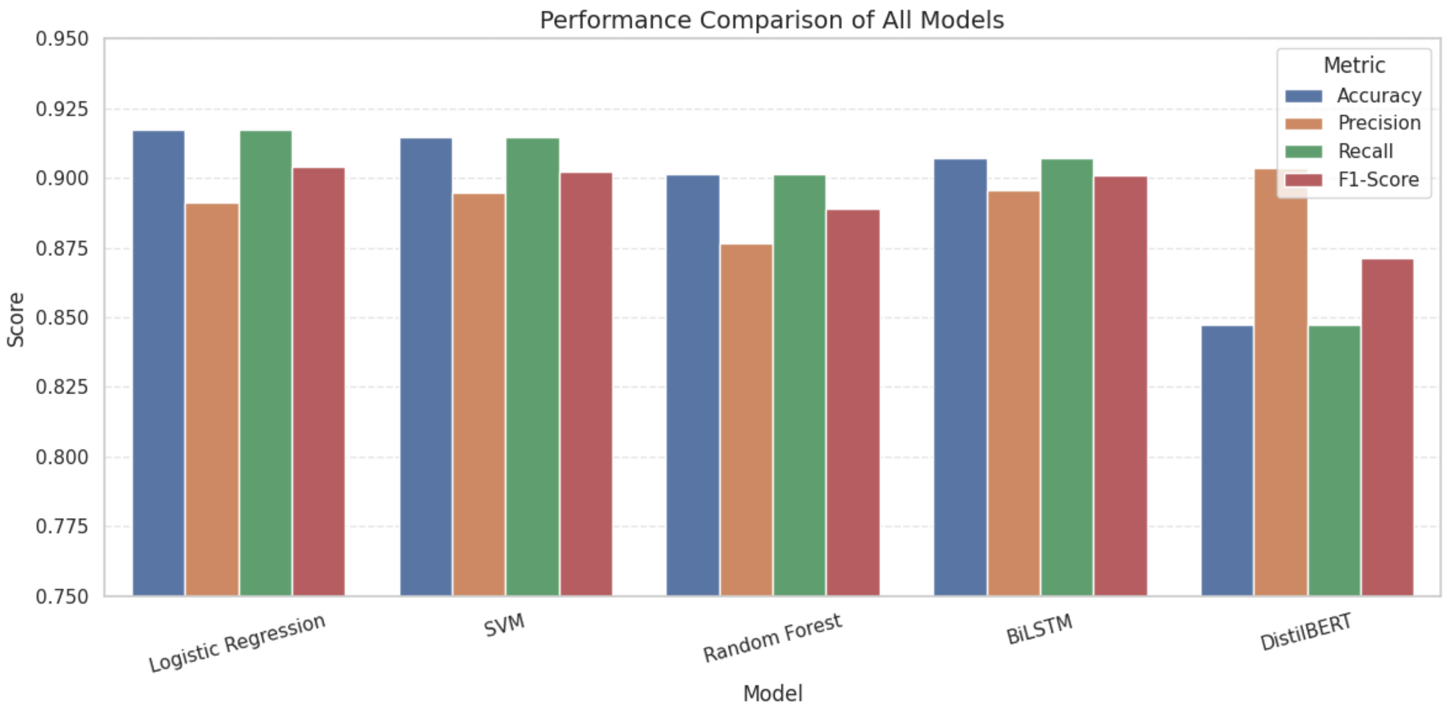


Figure 8.2 Matrix evaluation comparison

Table 8.1 Model Result

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.9173	0.8914	0.9173	0.9041
SVM	0.9145	0.8947	0.9145	0.9025
Random Forest	0.9016	0.8768	0.9016	0.8890
BiLSTM	0.9072	0.8955	0.9072	0.9008
DistilBERT	0.8475	0.9035	0.8475	0.8714

8.3 WordCloud Analysis by Sentiment

To better understand the language users employ in their reviews, word clouds were generated for each sentiment category. These images highlight the most common words. We distinguish it into 3 categories which had mentioned in Chapter 8.1, positive, neutral and negative.

Positive review, Words like "good", "driver", "service", and "thank" frequently appear, indicating satisfaction with drivers and service quality. Figure 8.3 Top Words in Positive Reviews

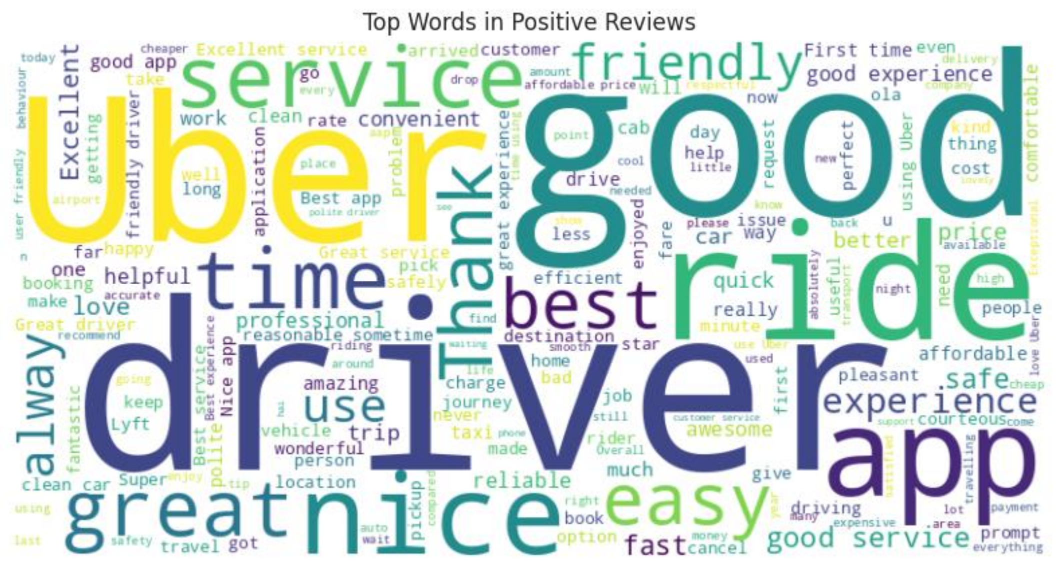


Figure 8.3 Positive WordCloud

Neutral reviews include descriptive or transactional language like "price," "location," and "option.". Figure 8.4 showing the top words in neutral reviews

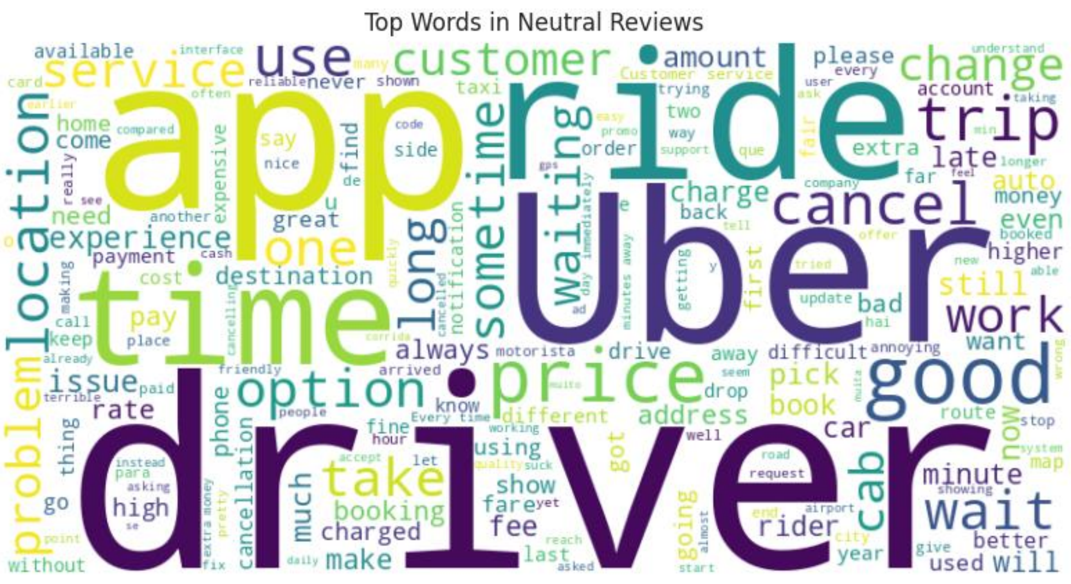


Figure 8.4 Neutral WordCloud

Negative reviews highlight problems such as "cancel," "wait," "price," "worst app," and "customer support," reflecting dissatisfaction with the reliability of the application and its customer services. Figure 8.5 showing the top words in negative reviews

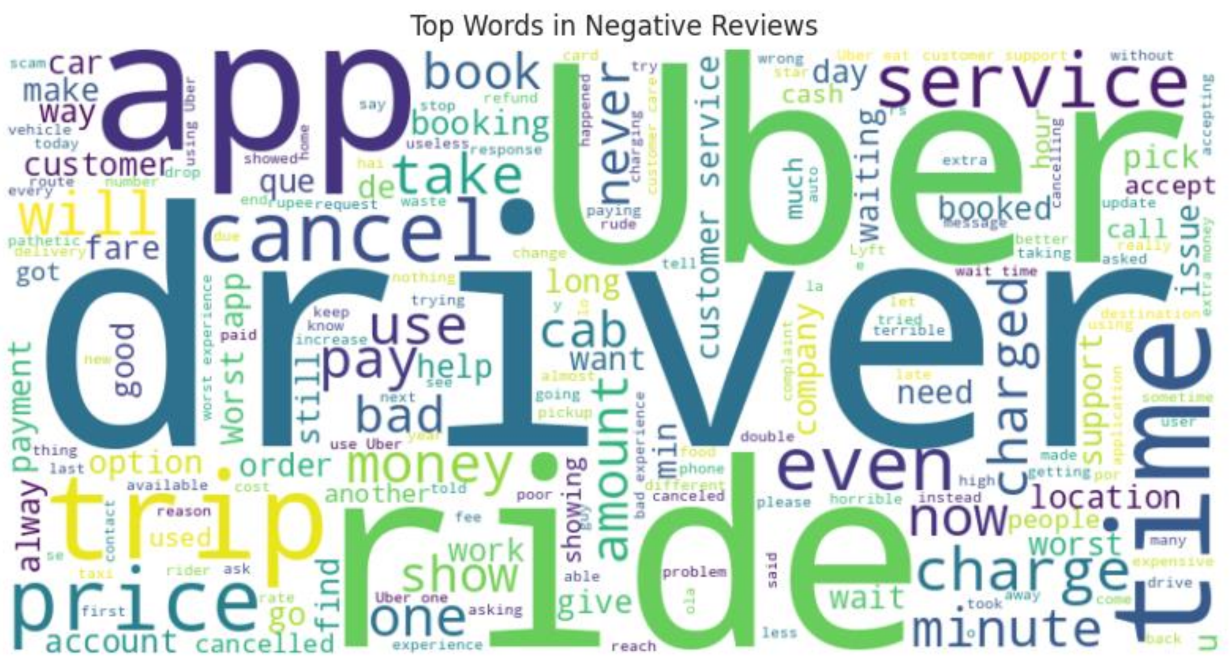
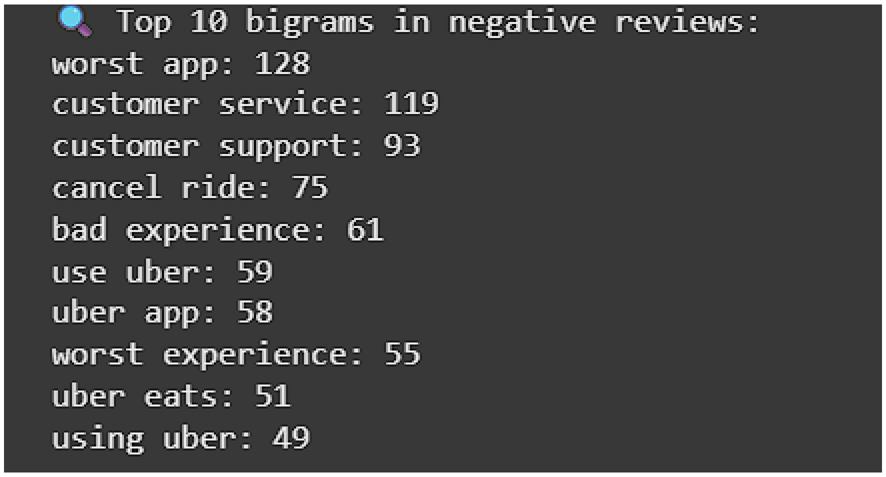


Figure 8.5 Nagative WordCloud

8.4 Bigram Frequency in Negative Reviews

To delve deeper into negative sentiment, bigrams (two-word combinations) were analyzed. The top 10 most frequent bigrams are shown in figure 8.6. Based on the negative review, business can determine which part need to have improvement and revision.



```
🔍 Top 10 bigrams in negative reviews:
worst app: 128
customer service: 119
customer support: 93
cancel ride: 75
bad experience: 61
use uber: 59
uber app: 58
worst experience: 55
uber eats: 51
using uber: 49
```

Figure 8.6 Negative Review Bigram Frequency

Based on the figure 8.6 result, the “worst app” (128 mentions) “customer service” (119 mentions) “cancel ride”, “use uber”, and “bad experience” show frequent use as well. These concerns focus on the performance of the application, reliability of services provided, and failures in achieving successful bookings.

The business insights alongside these visualizations are informative, as all capture model-driven business concerns that require actionable improvements. Beyond validating the model captures business problems within driving experiences, there are clear gaps within support responsiveness marked by pricing setbacks alongside high cancellation rates.

9. Discussion

The sentiments analysis of Uber customer feedback, presented in this report, provides valuable information on consumer sentiments and experiences related to the ride-sharing company. Through a rigorous process involving data collection, preprocessing, feature extraction, and applying varied machine learning and deep learning techniques, our aim was to identify customer sentiments correctly as positive, negative, or neutral.

Our findings show that extensive preprocessing of text data, including operations like tokenization, lowercasing, stop-word removal, and applying either stemming or lemmatization, was essential in making the unstructured raw review data suitable for successful model training. The feature engineering step, using methods like TF-IDF and possibly word embeddings, as indicated by the use of deep learning models, translated the textual data into numerical representations amenable for computational examination.

The exploration of both traditional machine learning methods and newer deep learning architectures, specifically those that make use of Transformer models, has shown that each method has unique strengths and weaknesses in the effective comprehension of customer sentiment nuances. Traditional models provide a reliable baseline and are more interpretable; deep learning models, however, show improved effectiveness due to their ability to detect broad patterns and contextual relationships within textual data, thus overcoming the complexity and semantic richness found in customer sentiment. The subtopic "Transformer and Deep Learning Models" would most probably detail specific implementations and their respective performance indicators (e.g., accuracy, precision, recall, and F1-score), thus shedding more light on these results. The ability of these models to capture subtle sentiment expressions, such as sarcasm or implied negativity, highlights their potential for competent sentiment categorization in practical applications.

The practical implications of these findings for Uber are significant. By accurately identifying sentiment trends, Uber can gain a deeper understanding of customer satisfaction levels, pinpoint specific areas requiring improvement (e.g., driver behavior, app functionality, pricing, wait times), and respond proactively to negative feedback. This data-driven approach can inform strategic decisions, enhance service quality, and ultimately foster a more positive customer experience. For instance, a surge in negative sentiment related to "long wait times" could prompt Uber to optimize driver allocation or introduce incentives in specific areas.

However, some limitations are warrant consideration. The validity and representativeness of the dataset collected are paramount; data biases, e.g., overrepresentation of extreme opinions or particular demographic groups, can affect the model's effectiveness in generalizing. The dynamic nature of online reviews means that sentiments and dominant discourses can shift over time, thus necessitating constant retraining and fine-tuning of the model. Further, the interpretability of complex deep learning models presents a major hindrance, making it difficult to understand the justification behind particular sentiment predictions. Although quantitative measures make it possible to make measurable performance assessments, a qualitative analysis of the decision process behind the model is often needed to derive actionable business insights.

10. Conclusion and Future Work

This project has successfully developed and evaluated a sentiment analysis model for customer reviews for Uber, demonstrating the efficacy of both machine learning and deep learning approaches in classifying consumer sentiments. A step-by-step methodology was developed, covering processes from data collection and preprocessing to model training and testing, which provides a solid basis for analyzing customer reviews. The use of advanced models, particularly those based on deep learning approaches, was instrumental in achieving greater classification accuracy, thereby facilitating Uber's ability to glean meaningful insights from the large amount of user-generated data. Overall, this study adds to the use of social media data for business intelligence and customer relationship management in the ride-sharing industry.

Future Work

Building upon the foundation laid by this study, several avenues for future research and development can be explored:

- **Aspect-Based Sentiment Analysis (ABSA):** Extend the current sentiment analysis to an aspect-based approach. Instead of merely classifying overall review sentiment, ABSA would identify specific aspects or features of Uber's service and determine the sentiment expressed towards each aspect. This would provide more granular and targeted feedback for service improvements.

- **Integration with Operational Data:** Integrate the results of sentiment analysis with Uber's operational data to allow for comparisons between customer sentiment and specific operational measures, thus providing a unified view of service performance.

- **Comparative Analysis with Other Ride-Sharing Services:** Conduct a comparative sentiment analysis of Uber with its competitors using the same methodological framework. This analysis can reveal competitive strengths or identify areas where Uber can learn from its competitors, as suggested by previous research comparing Uber and Ola.

11. References

[1] Md. Saymon Ahammad, S. A. Sinthia, Md. Muaj Chowdhury, N.-A.-A. Asif, and Md. Nurul AfsarIkram, “Sentiment Analysis of Various Ride Sharing Applications Reviews: A Comparative Analysis Between Deep Learning and Machine Learning Algorithms,” 2024, pp. 434–448. doi: 10.1007/978-3-031-69986-3_33.

[2] L. ALI, T. C. Gun, and W. Alhasan, “Comparative Analysis of Machine Learning Algorithms in Enhancing Healthcare Outcomes,” *European Modern Studies Journal*, vol. 8, no. 3, pp. 606–618, Jul. 2024, doi: 10.59573/emsj.8(3).2024.38.

[3] J. Anthal, A. Upadhyay, Y. Indulkar, and A. Patil, “Twitter Sentimental Analysis & Algorithm Comparison for Uber & Ola Using ‘R,’” *International Journal of Future Generation Communication and Networking*, vol. 13, no. 1s, 2020.

[4] V. Atina and P. Srisuk, “Sentiment Analysis of Grab App Reviews with Machine Learning Approach,” *Proceeding of International Conference on Science, Health, And Technology*, pp. 395–402, Sep. 2024, doi: 10.47701/icohetech.v5i1.4218.

[5] Y. Indulkar and A. Patil, “Sentiment Analysis of Uber &amp; Ola using Deep Learning,” in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, IEEE, Sep. 2020, pp. 21–27. doi: 10.1109/ICOSEC49089.2020.9215429.

[6] M. S. Sayeed, V. Mohan, and K. S. Muthu, “BERT: A Review of Applications in Sentiment Analysis,” *HighTech and Innovation Journal*, vol. 4, no. 2, pp. 453–462, Jun. 2023, doi: 10.28991/HIJ-2023-04-02-015.

[7] Md. S. Mahmud, A. Jaman Bonny, U. Saha, M. Jahan, Z. F. Tuna, and A. Al Marouf, “Sentiment Analysis from User-Generated Reviews of Ride-Sharing Mobile Applications,” in *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, Mar. 2022, pp. 738–744. doi: 10.1109/ICCMC53470.2022.9753947.

Dataset:

Kanchana Karunarathna. (2024). Uber Customer Reviews Dataset (2024) [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/10248932