Sentiment Analysis of British Airways Customer Reviews Using TF-IDF, Word2Vec and LSTM

In this project, I perform sentiment analysis on British Airways customer reviews using a comprehensive NLP pipeline that combines both traditional and deep learning techniques. The process begins with data loading and exploratory analysis to understand class distribution and text characteristics. I then apply preprocessing steps like lowercasing, punctuation removal, tokenization, stopword removal, and lemmatization. For feature extraction, I use both TF-IDF and Word2Vec representations and train classical machine learning models such as Logistic Regression and SVM. Building on this, I implement LSTM-based deep learning models—including a simple LSTM, bidirectional LSTM, stacked LSTM, and attention-based LSTM—to capture the sequential nature of text data. Each model is evaluated with metrics like accuracy, classification reports, and confusion matrices to compare performance. This end-to-end approach showcases the progression from foundational NLP methods to advanced neural architectures for sentiment classification.

Step 1

In the first step of my project, I begin by importing essential Python libraries required for data analysis and visualization, including pandas for data manipulation, NumPy for numerical operations, Matplotlib and Seaborn for plotting, and WordCloud for visualizing the most frequent words in the reviews. I then upload the dataset manually through Google Colab using files.upload(), allowing me to select the British Airways review file (BA_AirlineReviews.csv). Once the dataset is uploaded, I load it into a pandas DataFrame and display the first few rows using df.head() to verify that the data has been correctly imported and to get an initial overview of the structure and contents of the dataset.

```
# Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Install
!pip install wordcloud --quiet
```

```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
```

```
# Load dataset
from google.colab import files
uploaded = files.upload()  # Upload your BA_AirlineReviews.csv here manually

# Read CSV
df = pd.read_csv('BA_AirlineReviews.csv')

# Quick look
df.head()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving BA_AirlineReviews.csv to BA_AirlineReviews.csv
```

	Unnamed: 0	OverallRating	ReviewHeader	Name	Datetime	VerifiedReview	ReviewBody	TypeOfTraveller	SeatType	Route	DateFlown	SeatComfort	CabinStaffService	GroundS
0	0	1.0	"Service level far worse then Ryanair"	L Keele	19th November 2023	True	4 Hours before takeoff we received a Mail stat...	Couple Leisure	Economy Class	London to Stuttgart	November 2023	1.0	1.0	
1	1	3.0	"do not upgrade members based on status"	Austin Jones	19th November 2023	True	I recently had a delay on British Airways from...	Business	Economy Class	Brussels to London	November 2023	2.0	3.0	
2	2	8.0	"Flight was smooth and quick"	M A Collie	16th November 2023	False	Boarded on time, but it took ages to get to th...	Couple Leisure	Business Class	London Heathrow to Dublin	November 2023	3.0	3.0	
3	3	1.0	"Absolutely hopeless airline"	Nigel Dean	16th November 2023	True	5 days before the flight, we were advised by B...	Couple Leisure	Economy Class	London to Dublin	December 2022	3.0	3.0	
4	4	1.0	"Customer Service is non existent"	Gaylynne Simpson	14th November 2023	False	We traveled to Lisbon for our dream vacation, ...	Couple Leisure	Economy Class	London to Lisbon	November 2023	1.0	1.0	

Data preprocessing in NLP involves cleaning and transforming raw text into a structured format suitable for modeling. This typically includes steps like lowercasing, removing punctuation, tokenization, stopword removal, and lemmatization to improve model performance and reduce noise.

Examine the dataset's structure using df.info() to understand the data types and total entries, followed by checking for any missing values with df.isnull().sum(). I also assess the class distribution in the 'Recommended' column to see how balanced the labels are, which is important for model training and evaluation.

```
# Check basic info
print("Dataset Info:")
df.info()
```

```
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Check class balance
print("\nClass Distribution (Recommended Column):")
print(df['Recommended'].value_counts())
```

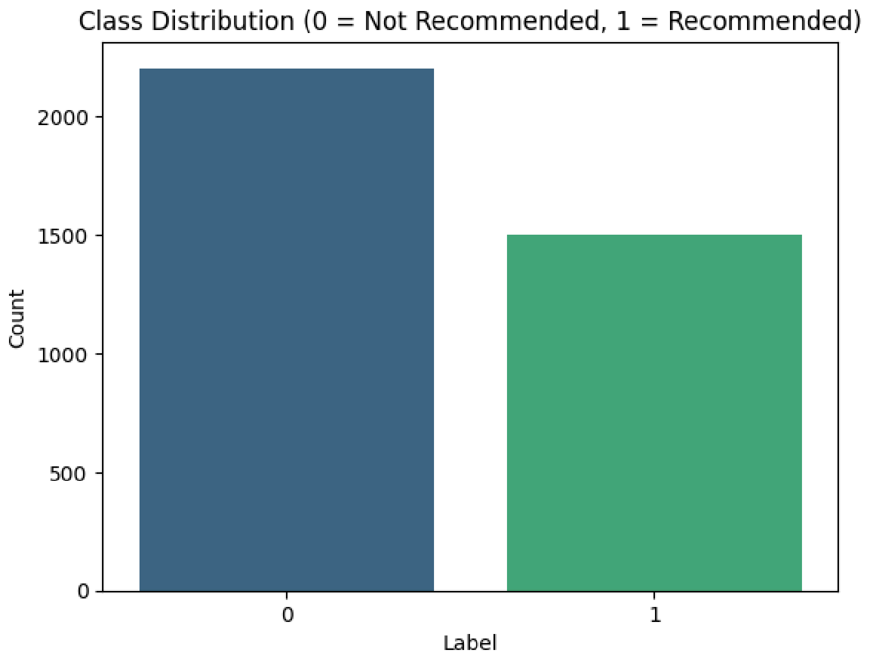
```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3701 entries, 0 to 3700
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            3701 non-null   int64
 1   OverallRating         3696 non-null   float64
 2   ReviewHeader          3701 non-null   object
 3   Name                  3701 non-null   object
 4   Datetime              3701 non-null   object
 5   VerifiedReview        3701 non-null   bool
 6   ReviewBody            3701 non-null   object
 7   TypeOfTraveller       2930 non-null   object
 8   SeatType              3699 non-null   object
 9   Route                 2926 non-null   object
 10  DateFlown             2923 non-null   object
 11  SeatComfort           3585 non-null   float64
 12  CabinStaffService     3574 non-null   float64
 13  GroundService         2855 non-null   float64
 14  ValueForMoney         3700 non-null   float64
 15  Recommended           3701 non-null   object
 16  Aircraft              1922 non-null   object
 17  Food&Beverages        3315 non-null   float64
 18  InflightEntertainment  2551 non-null   float64
 19  Wifi&Connectivity     609 non-null    float64
dtypes: bool(1), float64(8), int64(1), object(10)
memory usage: 553.1+ KB

Missing Values:
Unnamed: 0                  0
OverallRating               5
ReviewHeader                0
Name                        0
Datetime                    0
VerifiedReview              0
ReviewBody                  0
TypeOfTraveller           771
SeatType                    2
Route                     775
DateFlown                 778
SeatComfort               116
CabinStaffService         127
GroundService             846
ValueForMoney               1
Recommended                 0
Aircraft                 1779
Food&Beverages            386
InflightEntertainment    1150
Wifi&Connectivity        3092
dtype: int64

Class Distribution (Recommended Column):
Recommended
no     2203
yes    1498
Name: count, dtype: int64
```

Double-click (or enter) to edit

Bar Chart:I plot a countplot using Seaborn to visualize the distribution of the newly created label column. This helps me quickly identify if the dataset is balanced or skewed between recommended and not recommended reviews.

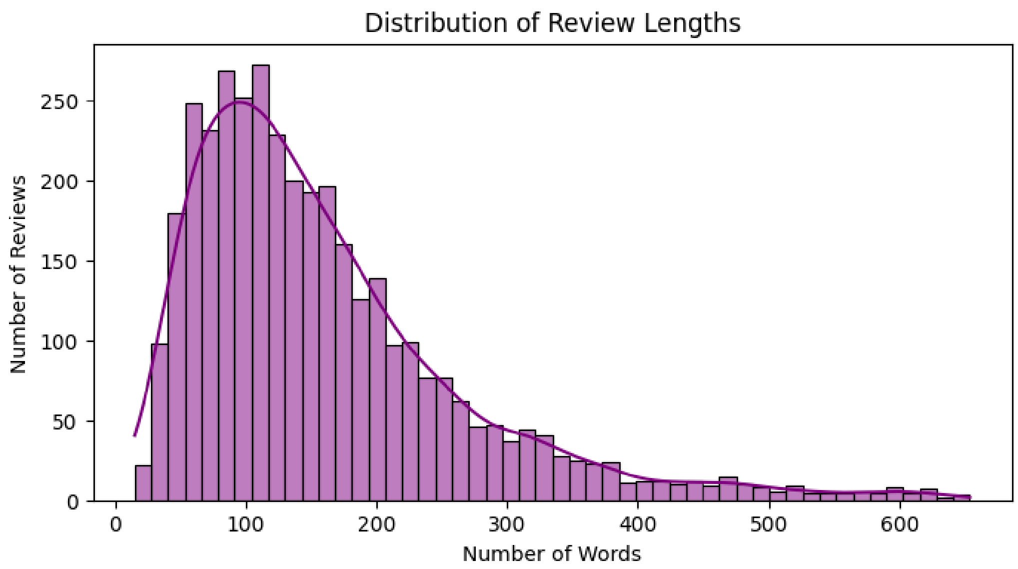
```
# Map 'yes'/'no' to 1/0
df['label'] = df['Recommended'].apply(lambda x: 1 if str(x).strip().lower() == 'yes' else 0)

# Plot
sns.countplot(x='label', data=df, palette='viridis')
plt.title('Class Distribution (0 = Not Recommended, 1 = Recommended)')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-5-af13da750281>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='label', data=df, palette='viridis')
```



Histogram of Review Lengths: I compute the number of words in each review and create a histogram to examine the distribution of review lengths. This gives insight into whether the reviews are mostly short or long, which can influence model design.

```
# Add review length column
df['review_length'] = df['ReviewBody'].apply(lambda x: len(str(x).split()))

# Plot
plt.figure(figsize=(8, 4))
sns.histplot(df['review_length'], bins=50, kde=True, color='purple')
plt.title('Distribution of Review Lengths')
plt.xlabel('Number of Words')
plt.ylabel('Number of Reviews')
plt.show()
```



Statistics Table: Using groupby, I generate descriptive statistics (count, mean, median, min, max) for review lengths in each class. This summary provides a quantitative comparison between recommended and not recommended review groups.

```
# Basic stats by class (Recommended / Not Recommended)
stats_table = df.groupby('label')['review_length'].agg(['count', 'mean', 'median', 'min', 'max']).reset_index()

# Map label for readability
stats_table['label'] = stats_table['label'].map({0: 'Not Recommended', 1: 'Recommended'})

# Display
print("📊 Review Length Statistics by Class:")
display(stats_table)
```

📊 Review Length Statistics by Class:

	label	count	mean	median	min	max
0	Not Recommended	2203	177.941443	151.0	19	654
1	Recommended	1498	134.043391	112.0	15	627

Exploratory Data Analysis (EDA) in NLP (in context of your coursework): In this coursework, EDA helps uncover key patterns and insights in the airline reviews before applying machine learning. It allows us to visualize class balance, examine text length variations, and understand frequent word usage in positive vs. negative reviews—ultimately guiding model selection and feature engineering.

```
# Install missing packages if needed
!pip install wordcloud --quiet
```

Word Clouds: I created separate word clouds for recommended and not recommended reviews to visually highlight the most frequent words in each sentiment class, giving intuitive insight into vocabulary differences.

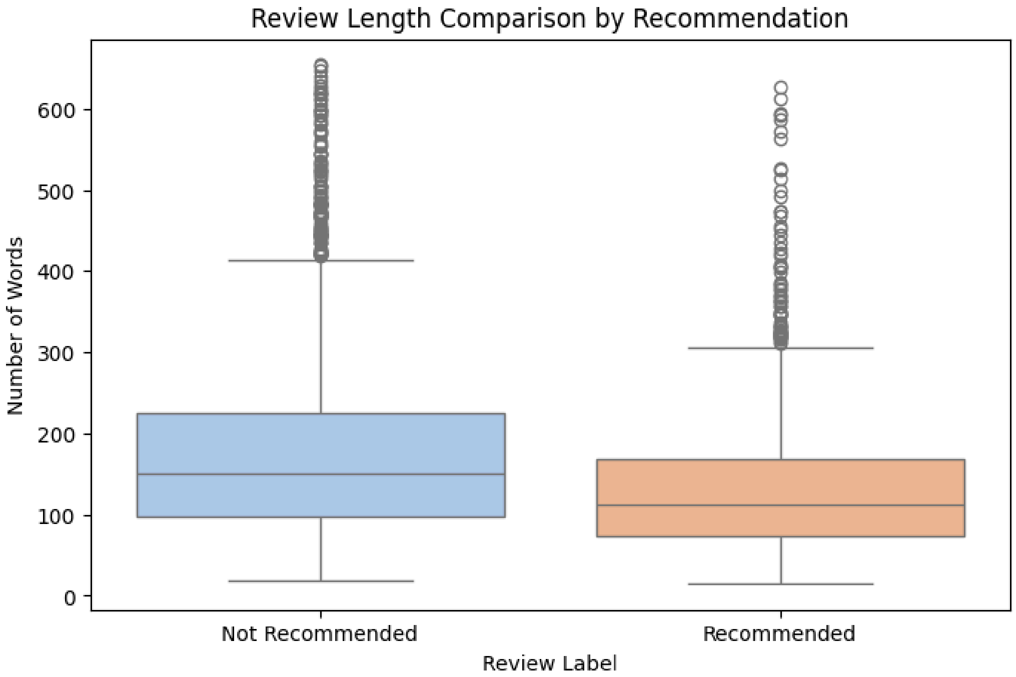
```
# Word cloud for Recommended Reviews
positive_reviews = " ".join(df[df['label']==1]['ReviewBody'].astype(str))
```

```
negative_reviews = " ".join(df[df['label']==0]['ReviewBody'].astype(str))

# Positive
plt.figure(figsize=(10,5))
wordcloud_pos = WordCloud(width=800, height=400, background_color='white').generate(positive_reviews)
plt.imshow(wordcloud_pos, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud: Recommended Reviews')
plt.show()

# Negative
plt.figure(figsize=(10,5))
wordcloud_neg = WordCloud(width=800, height=400, background_color='black', colormap='Reds').generate(negative_reviews)
plt.imshow(wordcloud_neg, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud: Not Recommended Reviews')
plt.show()
```

Box Plot of Review Lengths: The box plot compares the distribution of review lengths by label, revealing whether longer or shorter reviews tend to be associated with positive or negative feedback.

```
# Compare distributions visually
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x='label', y='review_length', palette='pastel')
plt.xticks([0,1], ['Not Recommended', 'Recommended'])
plt.title('Review Length Comparison by Recommendation')
plt.xlabel('Review Label')
plt.ylabel('Number of Words')
plt.show()
```

```
<ipython-input-9-10af8c140802>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x='label', y='review_length', palette='pastel')
```



Tokenization In this step of my coursework, I apply tokenization and text normalization to clean and prepare the British Airways reviews for model training. Tokenization splits text into individual words (tokens), enabling further analysis like lemmatization and stopword removal. This process is critical for converting raw text into structured, model-readable input. It ensures consistency and reduces noise, improving the performance of both traditional and deep learning models.

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

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

def preprocess_text(text):
    # Lowercase
    text = text.lower()
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    # Rejoin into cleaned sentence
    cleaned_text = " ".join(tokens)
    return cleaned_text
```

The code installs necessary NLP libraries (NLTK) and downloads required resources like stopwords and tokenizers. It defines a custom preprocessing function that lowercases text, removes punctuation and digits, tokenizes, removes stopwords, and lemmatizes words. The cleaned text is then stored in a new column called 'cleaned_review', which will be used for vectorization and modeling.

```
# 1. Install and Import Libraries
!pip install --quiet nltk

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

import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
# Download the missing 'punkt_tab' resource
nltk.download('punkt_tab') # This line was added to fix the LookupError


from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# 2. Define Preprocessing Function
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
def preprocess_text(text):
    if pd.isna(text):
        return ""
    text = text.lower()                              # Lowercase
    text = re.sub(r'<.*?>', '', text)                 # Remove HTML tags
    text = text.translate(str.maketrans('', '', string.punctuation))  # Remove punctuation
    text = re.sub(r'\d+', '', text)                   # Remove numbers
    tokens = word_tokenize(text)                      # Tokenize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]  # Lemmatize and remove stopwords
    cleaned_text = " ".join(tokens)                   # Rejoin
    return cleaned_text

# 3. Apply Preprocessing to Your Data
# Assuming your dataset is already loaded as df
# df = pd.read_csv('BA_AirlineReviews.csv')   # If not already loaded

# If not already mapped, map recommended column
if 'label' not in df.columns:
    df['label'] = df['Recommended'].apply(lambda x: 1 if str(x).strip().lower() == 'yes' else 0)

# Apply cleaning
df['cleaned_review'] = df['ReviewBody'].astype(str).apply(preprocess_text)

# 4. Preview cleaned reviews
print("\nSample Cleaned Reviews:")
df[['ReviewBody', 'cleaned_review']].sample(5)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.

Sample Cleaned Reviews:
```

	ReviewBody	cleaned_review
858	Miami to Delhi via London. The BA business cl...	miami delhi via london ba business class flew ...
2210	My wife and I used Avios to get two return tic...	wife used avios get two return ticket london h...
2733	British Airways Economy on a Boeing 777, Londo...	british airway economy boeing london bangkok s...
732	Buenos Aires to London Heathrow rwturn. The ai...	buenos aire london heathrow rwturn aircraft ol...
2212	London Heathrow to Newark return. Having just ...	london heathrow newark return returned holiday...

Step 3: TF-IDF + SMOTE + Baseline Machine Learning Models

In this step of my coursework, I establish traditional machine learning baselines using TF-IDF vectorization and three popular classifiers: Logistic Regression, Support Vector Machine (SVM), and Random Forest. This stage is essential because it provides a reference point to evaluate the effectiveness of more complex models (like LSTM or BERT). TF-IDF transforms textual data into numerical features by capturing word importance, while SMOTE addresses class imbalance by synthetically generating samples for the minority class. This improves generalization and fairness across classes.

The cleaned reviews are first converted into TF-IDF vectors with a vocabulary size limit of 5000. I split the dataset using stratified sampling and apply SMOTE to rebalance the training data. Then, for each model (Logistic Regression, SVM, Random Forest), I perform hyperparameter tuning using GridSearchCV to find the best configuration. I evaluate performance using accuracy, classification reports, and confusion matrices—providing both quantitative and visual insights into how well each model performs on real-world review data.

```
# Step 3: Baseline Machine Learning Models (TF-IDF + SMOTE + Hyperparameter Tuning)

# 1. Install Required Packages
!pip install --quiet imbalanced-learn scikit-learn

# 2. Import Libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from imblearn.over_sampling import SMOTE

import matplotlib.pyplot as plt
import seaborn as sns

# 3. TF-IDF Vectorization

# Vectorize cleaned reviews
tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['cleaned_review'])
y = df['label']

# 4. Train-Test Split (Stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# 5. Apply SMOTE to Training Data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

print(f"Original training size: {X_train.shape}")
print(f"Resampled training size: {X_train_res.shape}")

# 6. Model Training with Hyperparameter Tuning

def train_and_evaluate_model(model, param_grid, model_name):
    global best_accuracy, best_model_name, best_model_instance

    grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train_res, y_train_res)
```

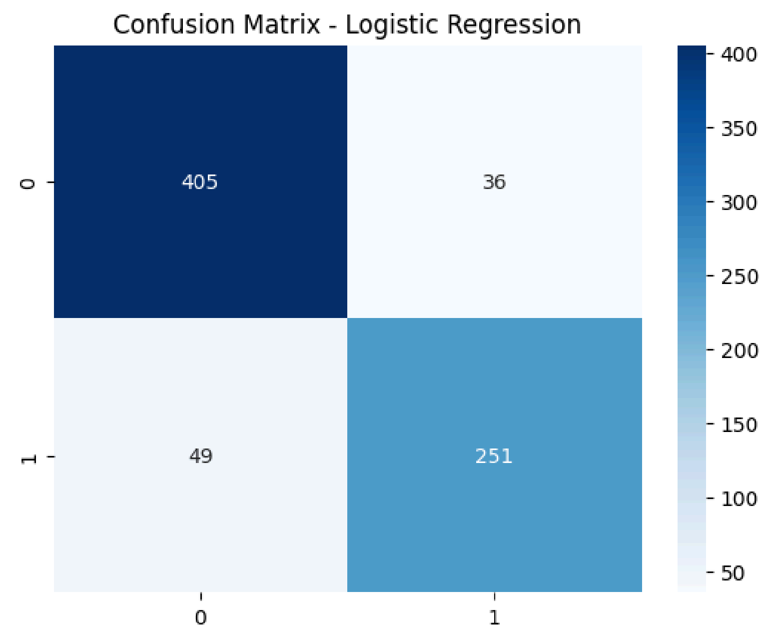
```
    best_model = grid.best_estimator_
    preds = best_model.predict(X_test)

    acc = accuracy_score(y_test, preds)
    print(f"\n\033[1m{model_name} (Best Parameters: {grid.best_params_})\033[0m")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, preds))
    sns.heatmap(confusion_matrix(y_test, preds), annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()
```

```
Original training size: (2960, 5000)
Resampled training size: (3524, 5000)
```

```
# Logistic Regression
lr_param_grid = {'C': [0.01, 0.1, 1, 10], 'penalty': ['l2']}
train_and_evaluate_model(LogisticRegression(max_iter=1000), lr_param_grid, 'Logistic Regression')

# SVM
svm_param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
train_and_evaluate_model(SVC(), svm_param_grid, 'Support Vector Machine')

# Random Forest
rf_param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 20, None]}
train_and_evaluate_model(RandomForestClassifier(), rf_param_grid, 'Random Forest')
```

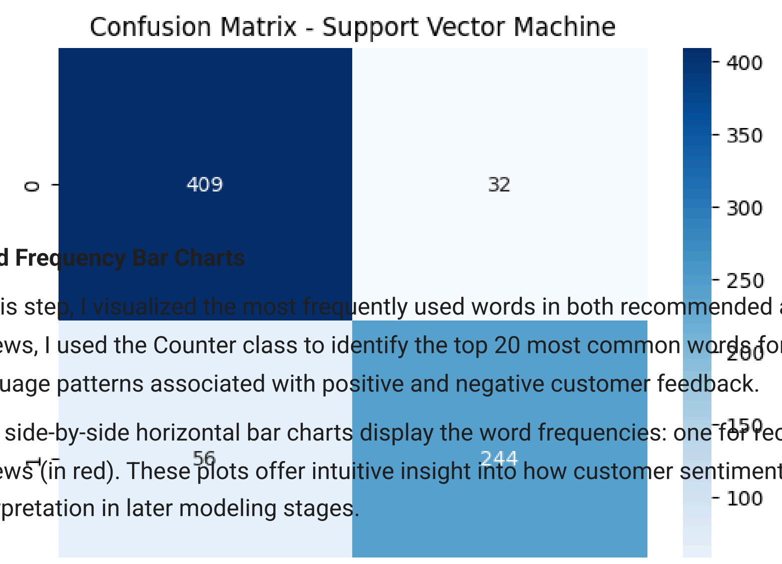
Logistic Regression (Best Parameters: {'C': 10, 'penalty': 'l2'})

```
Accuracy: 0.8853
              precision    recall  f1-score   support

           0       0.89      0.92      0.91       441
           1       0.87      0.84      0.86       300

    accuracy                           0.89       741
   macro avg       0.88      0.88      0.88       741
weighted avg       0.88      0.89      0.88       741
```



Support Vector Machine (Best Parameters: {'C': 10, 'kernel': 'rbf'})

```
Accuracy: 0.8812
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       441
           1       0.88      0.81      0.85       300

    accuracy                           0.88       741
   macro avg       0.88      0.87      0.88       741
weighted avg       0.88      0.88      0.88       741
```



Word Frequency Bar Charts

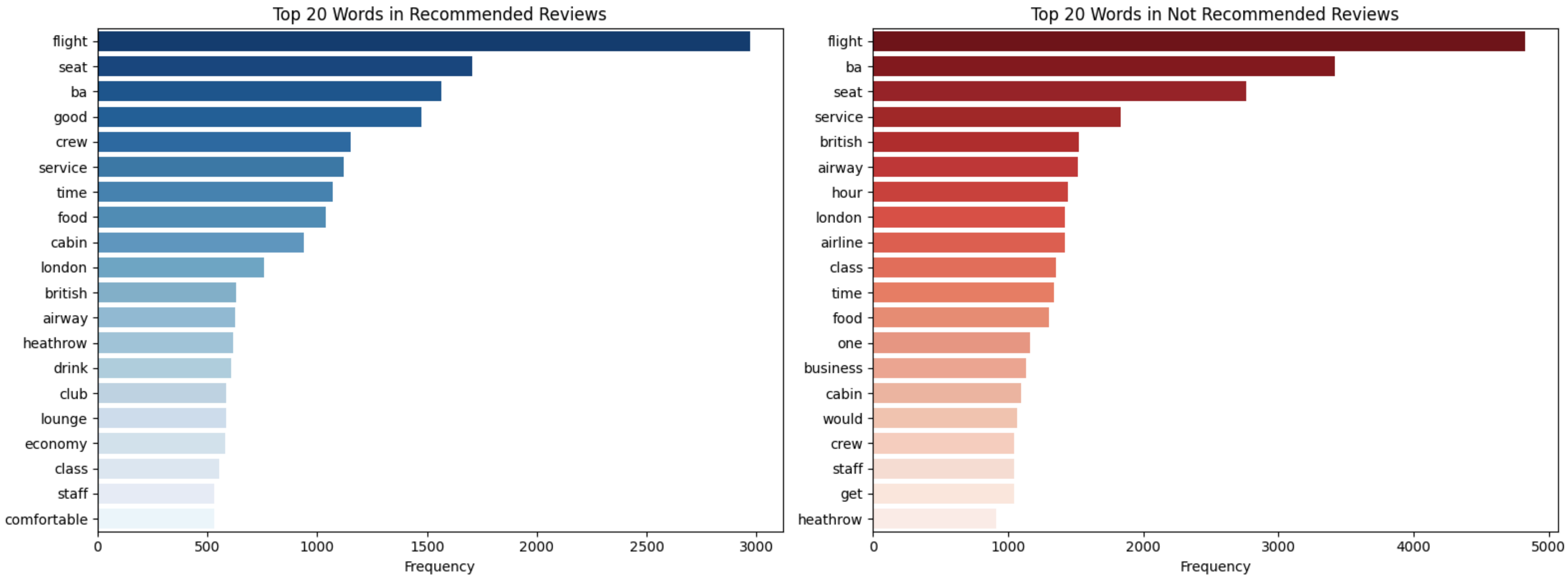
In this step, I visualized the most frequently used words in both recommended and not recommended reviews. After tokenizing the cleaned reviews, I used the Counter class to identify the top 20 most common words for each sentiment class. This analysis helps reveal the language patterns associated with positive and negative customer feedback.

Two side-by-side horizontal bar charts display the word frequencies: one for recommended reviews (in blue) and one for not recommended reviews (in red). These plots offer intuitive insight into how customer sentiment is expressed, which can support feature selection and interpretation in later modeling stages.

```
from collections import Counter
import nltk
nltk.download('punkt')

# Tokenize all words
positive_words = " ".join(df[df['label']==1]['cleaned_review'].dropna().astype(str)).split()
negative_words = " ".join(df[df['label']==0]['cleaned_review'].dropna().astype(str)).split()

# Top 20
top_pos = Counter(positive_words).most_common(20)
top_neg = Counter(negative_words).most_common(20)

# Plot
```

```
fig, axs = plt.subplots(1, 2, figsize=(16,6))

# Positive reviews
words_pos, freqs_pos = zip(*top_pos)
sns.barplot(x=list(freqs_pos), y=list(words_pos), ax=axs[0], palette="Blues_r")
axs[0].set_title('Top 20 Words in Recommended Reviews')
axs[0].set_xlabel('Frequency')

# Negative reviews
words_neg, freqs_neg = zip(*top_neg)
sns.barplot(x=list(freqs_neg), y=list(words_neg), ax=axs[1], palette="Reds_r")
axs[1].set_title('Top 20 Words in Not Recommended Reviews')
axs[1].set_xlabel('Frequency')

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
<ipython-input-32-4965837db32e>:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=list(freqs_pos), y=list(words_pos), ax=axs[0], palette="Blues_r")
<ipython-input-32-4965837db32e>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=list(freqs_neg), y=list(words_neg), ax=axs[1], palette="Reds_r")
```



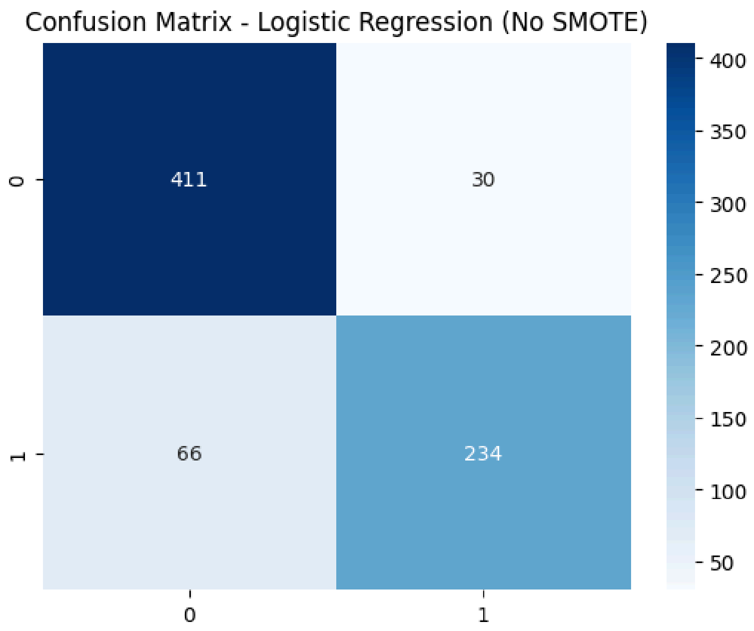
In this step, I trained and evaluated baseline machine learning models using TF-IDF features both with and without SMOTE (Synthetic Minority Oversampling Technique). SMOTE was applied because the dataset exhibited a class imbalance—more "Recommended" reviews than "Not Recommended"—which can bias models toward the majority class and degrade performance on the minority class.

By synthetically generating new examples of the underrepresented class in the training data, SMOTE helps balance the class distribution. This improves model generalization and ensures the classifier performs well on both positive and negative reviews. Comparing performance with and without SMOTE allows me to validate its impact and justify its use in improving recall and overall fairness of the models.

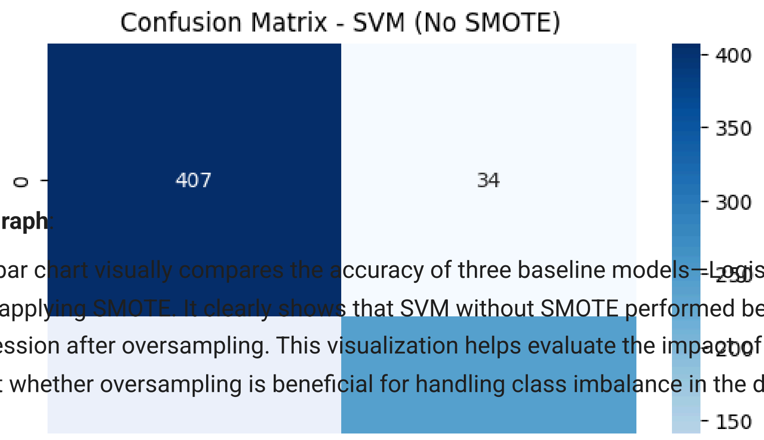
```
# Step 3: Baseline Machine Learning Models (TF-IDF) with and without SMOTE

# 1. Install Required Packages
!pip install --quiet imbalanced-learn scikit-learn

# 2. Import Libraries
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from imblearn.over_sampling import SMOTE

import matplotlib.pyplot as plt
import seaborn as sns

# 3. TF-IDF Vectorization

tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['cleaned_review'])
y = df['label']

# 4. Train-Test Split (Stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# 5. Helper Function to Train and Evaluate Models
def train_evaluate(model, X_train, y_train, X_test, y_test, model_name):
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f"\n\033[1m{model_name}\033[0m")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, preds))
    sns.heatmap(confusion_matrix(y_test, preds), annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()
```

```
# 6. Train Models without SMOTE
print("Training without SMOTE:")

train_evaluate(LogisticRegression(max_iter=1000), X_train, y_train, X_test, y_test, 'Logistic Regression (No SMOTE)')
train_evaluate(SVC(kernel='linear'), X_train, y_train, X_test, y_test, 'SVM (No SMOTE)')
train_evaluate(RandomForestClassifier(), X_train, y_train, X_test, y_test, 'Random Forest (No SMOTE)')

# 7. Apply SMOTE to Training Data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# 8. Train Models with SMOTE
print("\nTraining with SMOTE:")

train_evaluate(LogisticRegression(max_iter=1000), X_train_smote, y_train_smote, X_test, y_test, 'Logistic Regression (With SMOTE)')
train_evaluate(SVC(kernel='linear'), X_train_smote, y_train_smote, X_test, y_test, 'SVM (With SMOTE)')
train_evaluate(RandomForestClassifier(), X_train_smote, y_train_smote, X_test, y_test, 'Random Forest (With SMOTE)')
```

```
Training without SMOTE:

Logistic Regression (No SMOTE)
Accuracy: 0.8704
              precision    recall  f1-score   support

           0       0.86      0.93      0.90       441
           1       0.89      0.78      0.83       300

    accuracy                           0.87       741
   macro avg       0.87      0.86      0.86       741
weighted avg       0.87      0.87      0.87       741
```




```
SVM (No SMOTE)
Accuracy: 0.8785
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       441
           1       0.88      0.81      0.84       300

    accuracy                           0.88       741
   macro avg       0.88      0.87      0.87       741
weighted avg       0.88      0.88      0.88       741
```



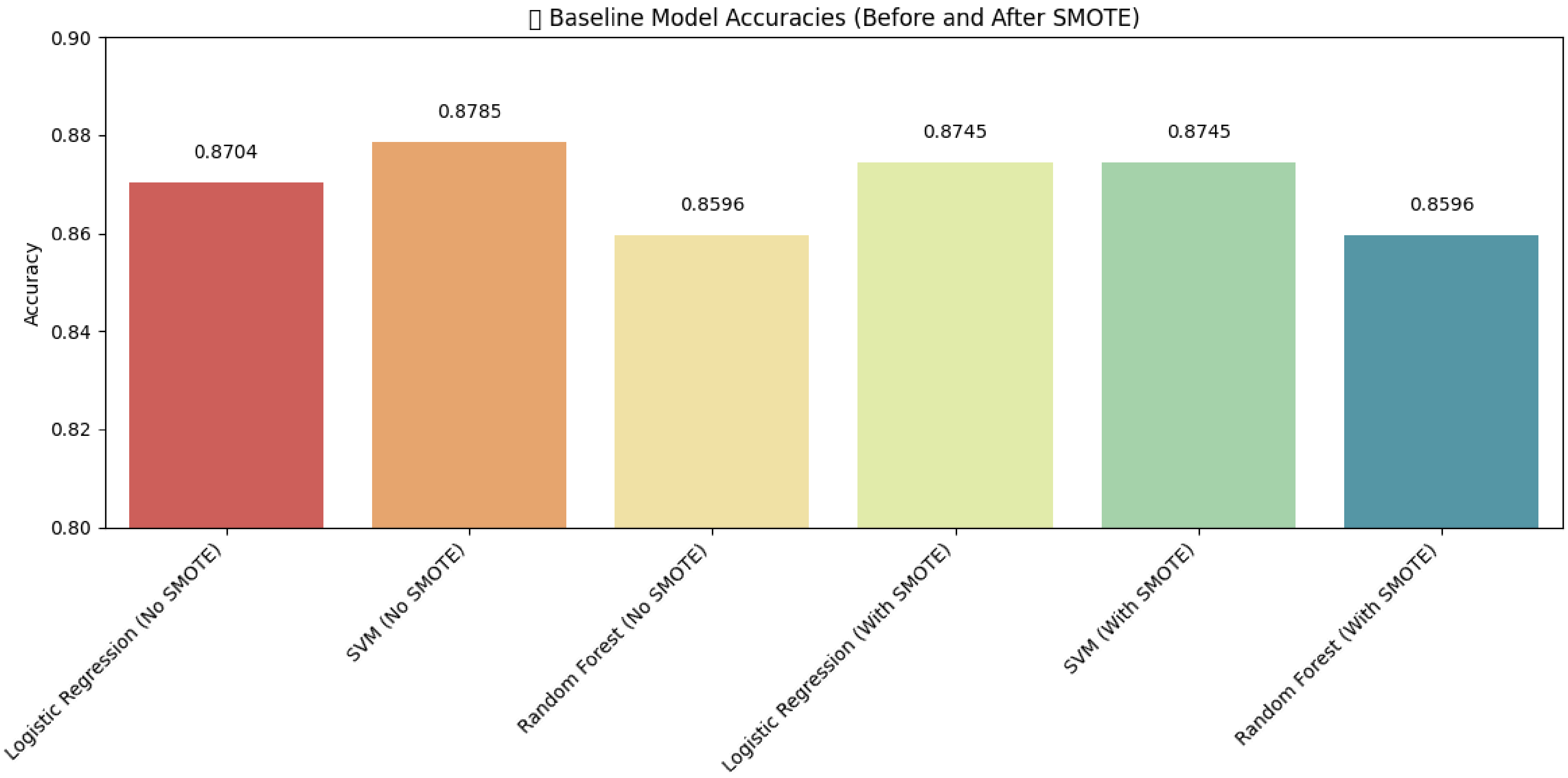
Bar Graph

This bar chart visually compares the accuracy of three baseline models—Logistic Regression, SVM, and Random Forest—both before and after applying SMOTE. It clearly shows that SVM without SMOTE performed best overall, with a slight improvement observed for Logistic Regression after oversampling. This visualization helps evaluate the impact of SMOTE on each model and supports data-driven decisions about whether oversampling is beneficial for handling class imbalance in the dataset.

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

# Accuracy results
model_names = [
    'Logistic Regression (No SMOTE)',
    'SVM (No SMOTE)',
    'Random Forest (No SMOTE)',
    'Logistic Regression (With SMOTE)',
    'SVM (With SMOTE)',
    'Random Forest (With SMOTE)'
]

accuracies = [
    0.8704,
    0.8785,
    0.8596,
    0.8745,
    0.8745,
    0.8596
]

# Plotting
plt.figure(figsize=(12,6))
sns.barplot(x=model_names, y=accuracies, palette="Spectral")
plt.xticks(rotation=45, ha='right')
```

```
plt.ylabel('Accuracy')
plt.title('📊 Baseline Model Accuracies (Before and After SMOTE)')
for index, value in enumerate(accuracies):
    plt.text(index, value + 0.005, f'{value:.4f}', ha='center', fontsize=10)
plt.ylim(0.8, 0.9)
plt.tight_layout()
plt.show()
```

```
<ipython-input-34-20b72ad0b369>:25: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=model_names, y=accuracies, palette="Spectral")
<ipython-input-34-20b72ad0b369>:32: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



Step 4:Word2Vec Embedding and Model Training

In Step 4 of my NLP coursework, I implemented Word2Vec to convert cleaned text reviews into dense vector representations that capture semantic meaning. Word2Vec is a powerful word embedding technique that helps in representing words in a continuous vector space, enabling machine learning models to better understand context and relationships between words.

What I Did in the Code:

1. **Tokenization for Word2Vec**: I split the cleaned review text into tokens (words) to prepare it for training the Word2Vec model.
2. **Training Word2Vec**: I used Gensim's `Word2Vec` with parameters such as `vector_size=100`, `window=5`, and `sg=1` (Skip-Gram) to learn embeddings from the reviews. Only words appearing at least twice (`min_count=2`) were considered.
3. **Review Vector Generation**: For each review, I averaged the vectors of all words in that review to create a fixed-length feature vector. This transformed the textual data into numerical form suitable for ML models.
4. **Model Training & Evaluation**: I trained and evaluated multiple classifiers (Logistic Regression, SVM, Random Forest, KNN, and Gradient Boosting) using the review vectors. Each model's accuracy and performance were visualized with a confusion matrix.
5. **Model Comparison Plot**: Finally, I plotted a bar chart comparing the accuracy of all Word2Vec-based models. This helps in understanding which algorithm performed best using semantic vector embeddings.

```
!pip install gensim --quiet
```

```
!pip install numpy==1.24.3
```

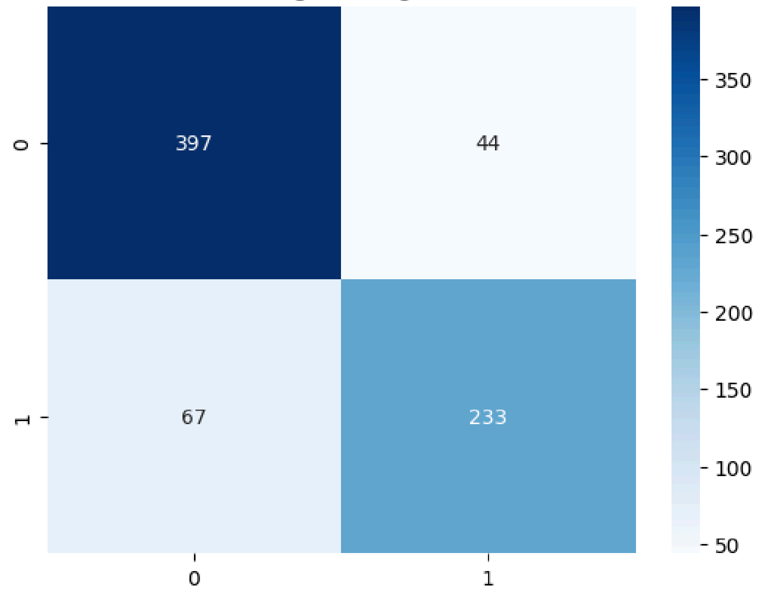
```
Requirement already satisfied: numpy==1.24.3 in /usr/local/lib/python3.11/dist-packages (1.24.3)
```

```
!pip install --quiet gensim scikit-learn
```

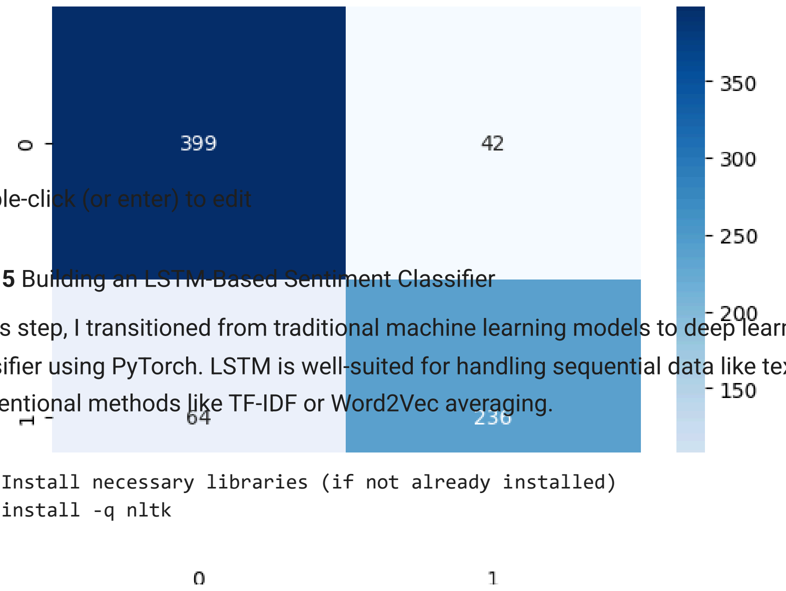
```
#Import Libraries
import gensim
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Prepare Tokenized Data for Word2Vec
# Assuming 'cleaned_review' column already exists from preprocessing step
tokenized_reviews = df['cleaned_review'].apply(lambda x: x.split())

# Train Word2Vec Model
w2v_model = Word2Vec(sentences=tokenized_reviews, vector_size=100, window=5, min_count=2, workers=4, sg=1)

# Create Feature Vectors for Each Review
```

```python
def get_review_vector(tokens, model, vector_size):
    vec = np.zeros(vector_size)
    count = 0
    for word in tokens:
        if word in model.wv.key_to_index:
            vec += model.wv[word]
            count += 1
    if count != 0:
        vec /= count
    return vec

vector_size = 100
review_vectors = np.array([get_review_vector(tokens, w2v_model, vector_size) for tokens in tokenized_reviews])

# Train-Test Split
X_train_w2v, X_test_w2v, y_train_w2v, y_test_w2v = train_test_split(
    review_vectors, df['label'], test_size=0.2, random_state=42, stratify=df['label'])

# Train and Evaluate Multiple ML Models
model_scores_w2v = {}

def train_evaluate_model_w2v(model, model_name):
    model.fit(X_train_w2v, y_train_w2v)
    preds = model.predict(X_test_w2v)
    acc = accuracy_score(y_test_w2v, preds)
    model_scores_w2v[model_name] = acc
    print(f"\n\033[1m{model_name}\033[0m")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test_w2v, preds))
    sns.heatmap(confusion_matrix(y_test_w2v, preds), annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()

# Logistic Regression
train_evaluate_model_w2v(LogisticRegression(max_iter=1000), 'Logistic Regression (Word2Vec)')

# SVM
train_evaluate_model_w2v(SVC(kernel='linear'), 'SVM (Word2Vec)')

# Random Forest
train_evaluate_model_w2v(RandomForestClassifier(n_estimators=100, random_state=42), 'Random Forest (Word2Vec)')

# K-Nearest Neighbors
train_evaluate_model_w2v(KNeighborsClassifier(n_neighbors=5), 'KNN (Word2Vec)')

# Gradient Boosting
train_evaluate_model_w2v(GradientBoostingClassifier(n_estimators=100, random_state=42), 'Gradient Boosting (Word2Vec)')

# 8. Plot Comparison of Word2Vec Models
plt.figure(figsize=(10,6))
sns.barplot(x=list(model_scores_w2v.keys()), y=list(model_scores_w2v.values()), palette='magma')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Accuracy')
plt.title('📊 Word2Vec-Based Model Accuracies')
for index, value in enumerate(model_scores_w2v.values()):
    plt.text(index, value + 0.01, f'{value:.4f}', ha='center', fontsize=10)
plt.ylim(0.5, 1)
plt.tight_layout()
plt.show()
```

```
Logistic Regression (Word2Vec)
Accuracy: 0.8502
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       441
           1       0.84      0.78      0.81       300

    accuracy                           0.85       741
   macro avg       0.85      0.84      0.84       741
weighted avg       0.85      0.85      0.85       741
```




```
SVM (Word2Vec)
Accuracy: 0.8570
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       441
           1       0.85      0.79      0.82       300

    accuracy                           0.86       741
   macro avg       0.86      0.85      0.85       741
weighted avg       0.86      0.86      0.86       741
```



Double-click (or enter) to edit

Step 5 Building an LSTM-Based Sentiment Classifier

In this step, I transitioned from traditional machine learning models to deep learning by building a custom LSTM (Long Short-Term Memory) classifier using PyTorch. LSTM is well-suited for handling sequential data like text, as it can capture contextual dependencies better than conventional methods like TF-IDF or Word2Vec averaging.

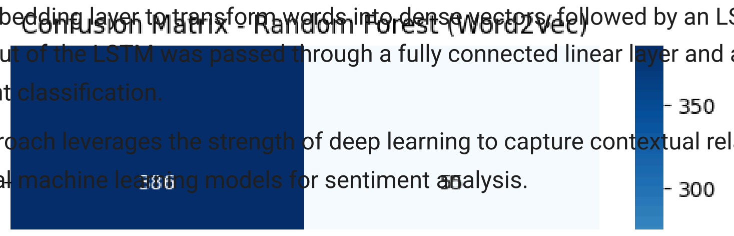
```
# 1. Install necessary libraries (if not already installed)
!pip install -q nltk
```

```
!pip install tensorflow --quiet
```

```
Random Forest (Word2Vec)
Accuracy: 0.8408
              precision    recall  f1-score   support

           0       0.86      0.88      0.87       441
           1       0.81      0.79      0.80       300

    accuracy                           0.84       741
   macro avg       0.84      0.83      0.83       741
weighted avg       0.84      0.84      0.84       741
```

In this step, we implemented a sentiment classification model using a Long Short-Term Memory (LSTM) neural network, which is highly effective for handling sequential text data. We began by preprocessing the reviews: converting text to lowercase, removing HTML tags, punctuation, and numbers, followed by tokenization, stopword removal, and lemmatization. This cleaned text was then tokenized using TensorFlow's Tokenizer and converted into padded sequences to ensure uniform input lengths.

After preprocessing, we prepared the data for PyTorch by converting it into tensors and organizing it into a custom Dataset class. This structure allowed us to efficiently feed batches of data into the model using PyTorch's DataLoader. The LSTM model architecture consisted of an embedding layer to transform words into dense vectors, followed by an LSTM layer that captures sequential dependencies in the text. The output of the LSTM was passed through a fully connected linear layer and a sigmoid activation function to produce binary predictions for sentiment classification.



This approach leverages the strength of deep learning to capture contextual relationships in text, offering a more advanced alternative to traditional machine learning models for sentiment analysis.

```
import pandas as pd
import numpy as np
import re
import string
import nltk
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
# Import Tokenizer from tensorflow.keras.preprocessing.text
from tensorflow.keras.preprocessing.text import Tokenizer # Changed import statement
from tensorflow.keras.utils import pad_sequences # Changed import statement
```

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
# 2. Load and prepare data (assuming df is already available)
# Ensure label column exists
if 'label' not in df.columns:
    df['label'] = df['Recommended'].apply(lambda x: 1 if str(x).strip().lower() == 'yes' else 0)
```

```
# Full Clean Step 2: Text Preprocessing

# 1. Install and Import Libraries
!pip install --quiet nltk

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

import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
# Download the missing 'punkt_tab' resource
nltk.download('punkt_tab') # This line was added to fix the LookupError


from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# 2. Define Preprocessing Function
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    if pd.isna(text):
        return ""
    text = text.lower()                              # Lowercase
    text = re.sub(r'<.*?>', '', text)                 # Remove HTML tags
    text = text.translate(str.maketrans('', '', string.punctuation))  # Remove punctuation
    text = re.sub(r'\d+', '', text)                   # Remove numbers
    tokens = word_tokenize(text)                      # Tokenize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]  # Lemmatize and remove stopwords
    cleaned_text = " ".join(tokens)                   # Rejoin
    return cleaned_text

# 3. Apply Preprocessing to Your Data
# Assuming your dataset is already loaded as df
# df = pd.read_csv('BA_AirlineReviews.csv')   # If not already loaded

# If not already mapped, map recommended column
if 'label' not in df.columns:
    df['label'] = df['Recommended'].apply(lambda x: 1 if str(x).strip().lower() == 'yes' else 0)

# Apply cleaning
df['cleaned_review'] = df['ReviewBody'].astype(str).apply(preprocess_text)

# 4. Preview cleaned reviews
print("\nSample Cleaned Reviews:")
df[['ReviewBody', 'cleaned_review']].sample(5)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!

Sample Cleaned Reviews:
```

	ReviewBody	cleaned_review
1002	Heathrow to Vancouver. The seats booked not gi...	heathrow vancouver seat booked given took mont...
1842	Gatwick to Barbados, outbound on 19th January ...	gatwick barbados outbound th january returning...
854	London to Dusseldorf. No free drinks or food, ...	london dusseldorf free drink food pretty norma...
2193	British Airways standards have dropped dramati...	british airway standard dropped dramatically f...
838	Return flight to Dublin. Outbound Galleries No...	return flight dublin outbound gallery north fi...

```
# 4. Tokenization and padding
MAX_VOCAB_SIZE = 10000
MAX_LEN = 100
tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, oov_token="<OOV>")
tokenizer.fit_on_texts(df['cleaned_review'])

X = tokenizer.texts_to_sequences(df['cleaned_review'])
X = pad_sequences(X, maxlen=MAX_LEN, padding='post')
y = df['label'].values
```

```
# 5. Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
# 6. Create Dataset class
class ReviewDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.long)
        self.y = torch.tensor(y, dtype=torch.float32)

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

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = ReviewDataset(X_train, y_train)
test_dataset = ReviewDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32)
```

Step 5 Building an LSTM-Based Sentiment Classifier

In this step, I transitioned from traditional machine learning models to deep learning by building a custom LSTM (Long Short-Term Memory) classifier using PyTorch. LSTM is well-suited for handling sequential data like text, as it can capture contextual dependencies better than conventional methods like TF-IDF or Word2Vec averaging.

Double-click (or enter) to edit

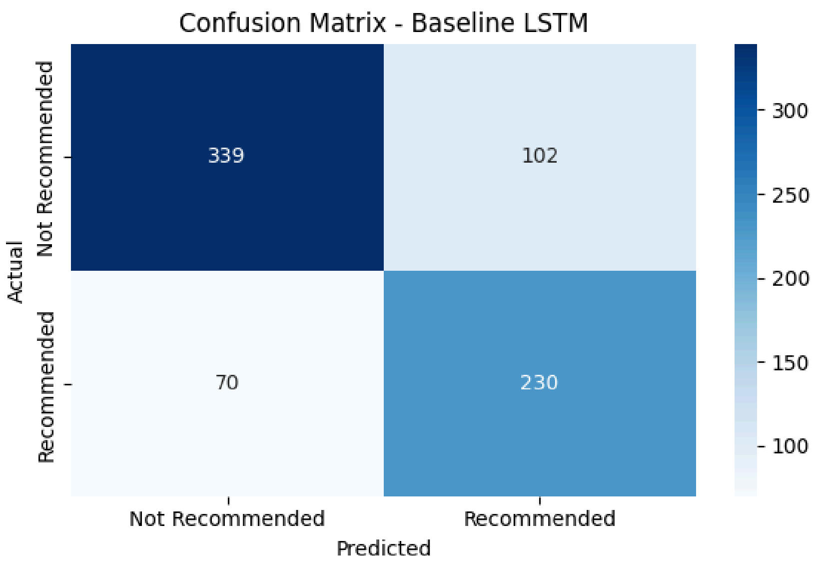
```
# 7. Define LSTM Model
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(LSTMClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        _, (hn, _) = self.lstm(x)
        out = self.fc(hn[-1])
        return self.sigmoid(out).squeeze()
```

Baseline Model :Implemented a baseline LSTM model for binary sentiment classification without dropout or tuning.Trained the model for 5 epochs and evaluated its performance using accuracy, classification report, and confusion matrix.This provides a foundational benchmark for comparison with more advanced deep learning models.

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Baseline LSTM Model (without tuning or dropout)

# Simple LSTM Classifier (no dropout, no bidirectional)
class BaselineLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(BaselineLSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        _, (hn, _) = self.lstm(x)
        out = self.fc(hn[-1])
        return self.sigmoid(out).squeeze()

# Initialize model
baseline_model = BaselineLSTM(vocab_size=MAX_VOCAB_SIZE, embedding_dim=100, hidden_dim=64)
criterion = nn.BCELoss()
optimizer = optim.Adam(baseline_model.parameters(), lr=0.001)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LSTMClassifier(vocab_size=MAX_VOCAB_SIZE, embedding_dim=128, hidden_dim=64)

model.to(device)

baseline_model.to(device)

# Train loop (basic, 5 epochs)
for epoch in range(5):
    baseline_model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        outputs = baseline_model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    print(f"[Baseline] Epoch {epoch+1}, Loss: {total_loss/len(train_loader):.4f}")

# Evaluate baseline
baseline_model.eval()
y_preds, y_true = [], []
with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
```

```
        outputs = baseline_model(batch_X)
        y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
        y_true.extend(batch_y.numpy().astype(int))

# Accuracy
baseline_accuracy = np.sum(np.array(y_preds) == np.array(y_true)) / len(y_true)
print(f"\nBaseline Model Accuracy: {baseline_accuracy:.4f}")

# Classification Report
print("\nClassification Report:\n")
print(classification_report(y_true, y_preds, target_names=["Not Recommended", "Recommended"]))

# Confusion Matrix
cm = confusion_matrix(y_true, y_preds)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Not Recommended", "Recommended"], yticklabels=["Not Recommended", "Recommended"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Baseline LSTM")
plt.tight_layout()
plt.show()
```

```
[Baseline] Epoch 1, Loss: 0.6827
[Baseline] Epoch 2, Loss: 0.6451
[Baseline] Epoch 3, Loss: 0.5711
[Baseline] Epoch 4, Loss: 0.4930
[Baseline] Epoch 5, Loss: 0.4078

Baseline Model Accuracy: 0.7679

Classification Report:

                 precision    recall  f1-score   support

Not Recommended       0.83      0.77      0.80       441
    Recommended       0.69      0.77      0.73       300

       accuracy                           0.77       741
      macro avg       0.76      0.77      0.76       741
   weighted avg       0.77      0.77      0.77       741
```

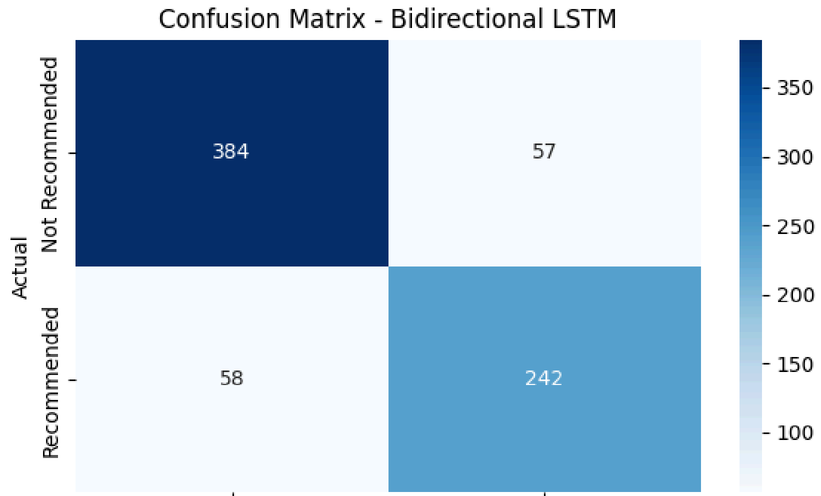


Bidirectional LSTM Developed an enhanced LSTM model using bidirectional layers, dropout for regularization, and early stopping to prevent overfitting.Trained the model for up to 10 epochs, selecting the best-performing version based on validation loss and test accuracy.Evaluated the final model with accuracy, classification report, and confusion matrix for performance assessment.

```
# Train the Model (with improvements: bidirectional LSTM, dropout, early stopping)

# Updated LSTM Classifier with bidirectional=True and dropout
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(LSTMClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(hidden_dim * 2, 1)  # *2 for bidirectional
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        pooled = torch.mean(lstm_out, dim=1)  # Average pooling
        out = self.dropout(pooled)
        out = self.fc(out)
        return self.sigmoid(out).squeeze()

# Initialize model
model = LSTMClassifier(vocab_size=MAX_VOCAB_SIZE, embedding_dim=128, hidden_dim=64)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Early stopping variables
best_loss = float('inf')
best_accuracy = 0.0
best_epoch = 0
early_stop_count = 0
patience = 2

# Train loop with early stopping
for epoch in range(10):
    model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
```

```
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(train_loader)
    print(f"Epoch {epoch+1}, Loss: {avg_loss:.4f}")

    # Evaluate on test set
    model.eval()
    y_preds, y_true = [], []
    with torch.no_grad():
        for batch_X, batch_y in test_loader:
            batch_X = batch_X.to(device)
            outputs = model(batch_X)
            y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
            y_true.extend(batch_y.numpy().astype(int))
    correct = np.sum(np.array(y_preds) == np.array(y_true))
    accuracy = correct / len(y_true)
    print(f"Test Accuracy: {accuracy:.4f}")

    # Save best model
    if avg_loss < best_loss:
        best_loss = avg_loss
        best_accuracy = accuracy
        best_epoch = epoch + 1
        early_stop_count = 0
        torch.save(model.state_dict(), "best_lstm_model.pt")
    else:
        early_stop_count += 1
        if early_stop_count >= patience:
            print("Early stopping triggered.")
            break

# Print best model info
print(f"\nBest Model Info:\nEpoch: {best_epoch}, Accuracy: {best_accuracy:.4f}, Parameters: {sum(p.numel() for p in model.parameters() if p.requires_grad)}")

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the best model before final evaluation
model.load_state_dict(torch.load("best_lstm_model.pt"))
model.eval()

# Final predictions and ground truths
y_preds, y_true = [], []
with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
        outputs = model(batch_X)
        y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
        y_true.extend(batch_y.numpy().astype(int))

# Calculate final accuracy
final_accuracy = np.sum(np.array(y_preds) == np.array(y_true)) / len(y_true)

# Print best model info again
print(f"\nBest Model Info:\nEpoch: {best_epoch}, Accuracy: {best_accuracy:.4f}, Final Accuracy: {final_accuracy:.4f}")
print(f"Trainable Parameters: {sum(p.numel() for p in model.parameters() if p.requires_grad)}")

# Classification report
print("\nClassification Report:\n")
print(classification_report(y_true, y_preds, target_names=["Not Recommended", "Recommended"]))

# Confusion matrix
cm = confusion_matrix(y_true, y_preds)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Not Recommended", "Recommended"],
            yticklabels=["Not Recommended", "Recommended"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Bidirectional LSTM")
plt.tight_layout()
plt.show()
```

```
Epoch 1, Loss: 0.6197
Test Accuracy: 0.7746
Epoch 2, Loss: 0.4147
Test Accuracy: 0.8165
Epoch 3, Loss: 0.3026
Test Accuracy: 0.8448
Epoch 4, Loss: 0.2306
Test Accuracy: 0.8421
Epoch 5, Loss: 0.2129
Test Accuracy: 0.8435
Epoch 6, Loss: 0.1697
Test Accuracy: 0.8556
Epoch 7, Loss: 0.1073
Test Accuracy: 0.8529
Epoch 8, Loss: 0.0744
Test Accuracy: 0.8502
Epoch 9, Loss: 0.0561
Test Accuracy: 0.8475
Epoch 10, Loss: 0.0325
Test Accuracy: 0.8448

Best Model Info:
Epoch: 10, Accuracy: 0.8448, Parameters: 1379457

Best Model Info:
Epoch: 10, Accuracy: 0.8448, Final Accuracy: 0.8448
Trainable Parameters: 1379457

Classification Report:

                 precision    recall  f1-score   support

Not Recommended       0.87      0.87      0.87       441
    Recommended       0.81      0.81      0.81       300

       accuracy                           0.84       741
      macro avg       0.84      0.84      0.84       741
   weighted avg       0.84      0.84      0.84       741
```

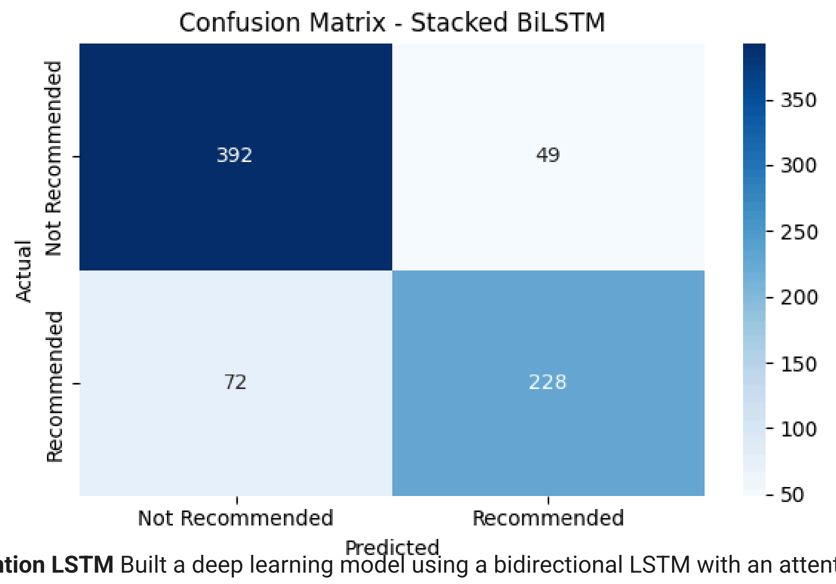


Stacked LSTM Implemented a deep learning NLP model using a two-layer bidirectional LSTM (Stacked BiLSTM) with dropout for regularization.Trained with early stopping and evaluated using test accuracy, classification metrics, and a confusion matrix.Captures complex sequential dependencies in text for binary classification of recommendation sentiment.

```
#Deep Learning NLP Model Using Stacked BiLSTM

class StackedLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(StackedLSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=2, batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(hidden_dim * 2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        pooled = torch.mean(lstm_out, dim=1)
        out = self.dropout(pooled)
        out = self.fc(out)
        return self.sigmoid(out).squeeze()

model = StackedLSTM(vocab_size=MAX_VOCAB_SIZE, embedding_dim=128, hidden_dim=64)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
model.to(device)

best_loss = float('inf')
best_accuracy = 0.0
best_epoch = 0
early_stop_count = 0
patience = 2

for epoch in range(10):
    model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(train_loader)
    print(f"[Stacked LSTM] Epoch {epoch+1}, Loss: {avg_loss:.4f}")

    model.eval()
    y_preds, y_true = [], []
    with torch.no_grad():
```

```
        for batch_X, batch_y in test_loader:
            batch_X = batch_X.to(device)
            outputs = model(batch_X)
            y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
            y_true.extend(batch_y.numpy().astype(int))
    correct = np.sum(np.array(y_preds) == np.array(y_true))
    accuracy = correct / len(y_true)
    print(f"[Stacked LSTM] Test Accuracy: {accuracy:.4f}")

    if avg_loss < best_loss:
        best_loss = avg_loss
        best_accuracy = accuracy
        best_epoch = epoch + 1
        early_stop_count = 0
        torch.save(model.state_dict(), "best_stacked_lstm.pt")
    else:
        early_stop_count += 1
        if early_stop_count >= patience:
            print("Early stopping triggered.")
            break

print(f"\nStacked LSTM:\nEpoch: {best_epoch}, Accuracy: {best_accuracy:.4f}, Parameters: {sum(p.numel() for p in model.parameters() if p.requires_grad)}")

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the best model for evaluation
model.load_state_dict(torch.load("best_stacked_lstm.pt"))
model.eval()

# Evaluate on test set
y_preds, y_true = [], []
with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
        outputs = model(batch_X)
        y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
        y_true.extend(batch_y.numpy().astype(int))

# Accuracy
accuracy = np.sum(np.array(y_preds) == np.array(y_true)) / len(y_true)
print(f"\nStacked BiLSTM Final Accuracy: {accuracy:.4f}")

# Classification Report
print("\nClassification Report:\n")
print(classification_report(y_true, y_preds, target_names=["Not Recommended", "Recommended"]))

# Confusion Matrix
cm = confusion_matrix(y_true, y_preds)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Not Recommended", "Recommended"],
            yticklabels=["Not Recommended", "Recommended"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Stacked BiLSTM")
plt.tight_layout()
plt.show()
```

```
[Stacked LSTM] Epoch 1, Loss: 0.5825
[Stacked LSTM] Test Accuracy: 0.7935
[Stacked LSTM] Epoch 2, Loss: 0.3656
[Stacked LSTM] Test Accuracy: 0.8246
[Stacked LSTM] Epoch 3, Loss: 0.2996
[Stacked LSTM] Test Accuracy: 0.8462
[Stacked LSTM] Epoch 4, Loss: 0.2169
[Stacked LSTM] Test Accuracy: 0.8529
[Stacked LSTM] Epoch 5, Loss: 0.1486
[Stacked LSTM] Test Accuracy: 0.8516
[Stacked LSTM] Epoch 6, Loss: 0.0919
[Stacked LSTM] Test Accuracy: 0.8502
[Stacked LSTM] Epoch 7, Loss: 0.0601
[Stacked LSTM] Test Accuracy: 0.8435
[Stacked LSTM] Epoch 8, Loss: 0.0630
[Stacked LSTM] Test Accuracy: 0.8502
[Stacked LSTM] Epoch 9, Loss: 0.0364
[Stacked LSTM] Test Accuracy: 0.8462
[Stacked LSTM] Epoch 10, Loss: 0.0211
[Stacked LSTM] Test Accuracy: 0.8367

Stacked LSTM:
Epoch: 10, Accuracy: 0.8367, Parameters: 1478785

Stacked BiLSTM Final Accuracy: 0.8367

Classification Report:

                 precision    recall  f1-score   support

Not Recommended       0.84      0.89      0.87       441
    Recommended       0.82      0.76      0.79       300

       accuracy                           0.84       741
      macro avg       0.83      0.82      0.83       741
   weighted avg       0.84      0.84      0.84       741
```

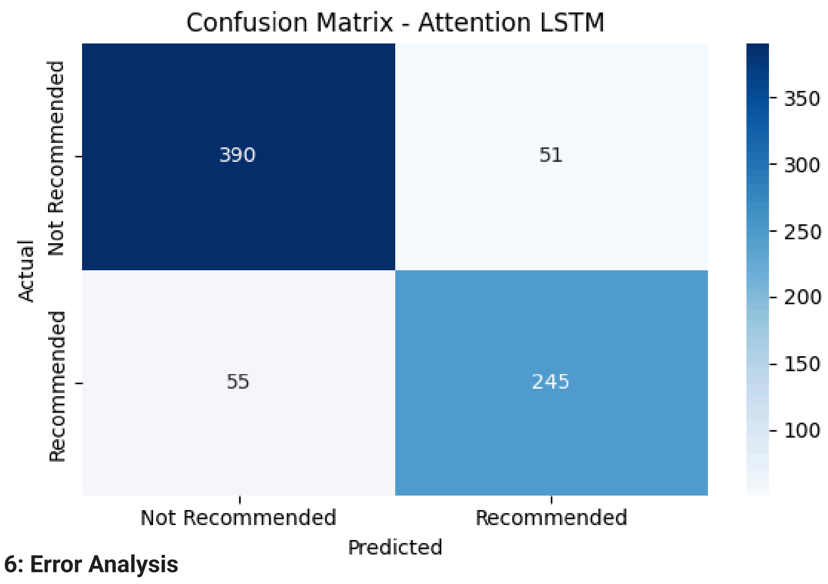


Attention LSTM Built a deep learning model using a bidirectional LSTM with an attention mechanism to focus on important parts of the input sequence.Integrated attention for context-aware feature extraction, improving interpretability and model focus. Evaluated using accuracy, classification metrics, and confusion matrix for binary recommendation classification.

```
# Deep Learning Model Using Attention LSTM (Single Layer)

import torch.nn.functional as F

class Attention(nn.Module):
    def __init__(self, hidden_dim):
        super(Attention, self).__init__()
        self.attn = nn.Linear(hidden_dim * 2, 1)

    def forward(self, lstm_out):
        weights = self.attn(lstm_out).squeeze(2)
        weights = F.softmax(weights, dim=1)
        context = torch.sum(lstm_out * weights.unsqueeze(2), dim=1)
        return context

class AttentionLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(AttentionLSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.attention = Attention(hidden_dim)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(hidden_dim * 2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        lstm_out, _ = self.lstm(x)
        attn_out = self.attention(lstm_out)
        out = self.dropout(attn_out)
        out = self.fc(out)
        return self.sigmoid(out).squeeze()

model = AttentionLSTM(vocab_size=MAX_VOCAB_SIZE, embedding_dim=128, hidden_dim=64)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
model.to(device)

best_loss = float('inf')
best_accuracy = 0.0
best_epoch = 0
early_stop_count = 0
patience = 2

for epoch in range(10):
    model.train()
    total_loss = 0
    for batch_X, batch_y in train_loader:
```

```
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(train_loader)
    print(f"[Attention LSTM] Epoch {epoch+1}, Loss: {avg_loss:.4f}")

    model.eval()
    y_preds, y_true = [], []
    with torch.no_grad():
        for batch_X, batch_y in test_loader:
            batch_X = batch_X.to(device)
            outputs = model(batch_X)
            y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
            y_true.extend(batch_y.numpy().astype(int))
    correct = np.sum(np.array(y_preds) == np.array(y_true))
    accuracy = correct / len(y_true)
    print(f"[Attention LSTM] Test Accuracy: {accuracy:.4f}")

    if avg_loss < best_loss:
        best_loss = avg_loss
        best_accuracy = accuracy
        best_epoch = epoch + 1
        early_stop_count = 0
        torch.save(model.state_dict(), "best_attention_lstm.pt")
    else:
        early_stop_count += 1
        if early_stop_count >= patience:
            print("Early stopping triggered.")
            break

print(f"\nAttention LSTM):\nEpoch: {best_epoch}, Accuracy: {best_accuracy:.4f}, Parameters: {sum(p.numel() for p in model.parameters() if p.requires_grad)}")

from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load best Attention LSTM model
model.load_state_dict(torch.load("best_attention_lstm.pt"))
model.eval()

# Final predictions and ground truths
y_preds, y_true = [], []
with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
        outputs = model(batch_X)
        y_preds.extend((outputs.cpu().numpy() > 0.5).astype(int))
        y_true.extend(batch_y.numpy().astype(int))

# Final accuracy
accuracy = np.sum(np.array(y_preds) == np.array(y_true)) / len(y_true)
print(f"\nAttention LSTM Final Accuracy: {accuracy:.4f}")

# Classification report
print("\nClassification Report:\n")
print(classification_report(y_true, y_preds, target_names=["Not Recommended", "Recommended"]))

# Confusion matrix
cm = confusion_matrix(y_true, y_preds)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Not Recommended", "Recommended"],
            yticklabels=["Not Recommended", "Recommended"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Attention LSTM")
plt.tight_layout()
plt.show()
```

```
[Attention LSTM] Epoch 1, Loss: 0.6039
[Attention LSTM] Test Accuracy: 0.8030
[Attention LSTM] Epoch 2, Loss: 0.3557
[Attention LSTM] Test Accuracy: 0.8408
[Attention LSTM] Epoch 3, Loss: 0.2340
[Attention LSTM] Test Accuracy: 0.8516
[Attention LSTM] Epoch 4, Loss: 0.1484
[Attention LSTM] Test Accuracy: 0.8596
[Attention LSTM] Epoch 5, Loss: 0.0956
[Attention LSTM] Test Accuracy: 0.8596
[Attention LSTM] Epoch 6, Loss: 0.0564
[Attention LSTM] Test Accuracy: 0.8650
[Attention LSTM] Epoch 7, Loss: 0.0419
[Attention LSTM] Test Accuracy: 0.8435
[Attention LSTM] Epoch 8, Loss: 0.0543
[Attention LSTM] Test Accuracy: 0.8408
[Attention LSTM] Epoch 9, Loss: 0.0375
[Attention LSTM] Test Accuracy: 0.8570
[Attention LSTM] Epoch 10, Loss: 0.0176
[Attention LSTM] Test Accuracy: 0.8570

Attention LSTM):
Epoch: 10, Accuracy: 0.8570, Parameters: 1379586

Attention LSTM Final Accuracy: 0.8570

Classification Report:

                 precision    recall  f1-score   support

Not Recommended       0.88      0.88      0.88       441
    Recommended       0.83      0.82      0.82       300

       accuracy                           0.86       741
      macro avg       0.85      0.85      0.85       741
   weighted avg       0.86      0.86      0.86       741
```



Step 6: Error Analysis

In this step, we analyze the misclassified outputs of the Stacked BiLSTM and Attention LSTM models to identify patterns and weaknesses in their predictions. By examining the confusion matrix and classification report, we can understand whether the models struggle more with "Recommended" or "Not Recommended" reviews. This helps uncover issues like class imbalance, ambiguous text, or insufficient contextual understanding. Such insights guide model improvements and data refinement.

Attention LSTM This step examines where the Attention LSTM model made incorrect predictions by decoding misclassified reviews. Common issues include ambiguity, sarcasm, or mixed sentiments that confuse the model. Analyzing these errors provides insights to refine data preprocessing and enhance model performance.

```
# Error Analysis for LSTM Model(Attention)

# Load best performing model (e.g., Attention LSTM)
model.load_state_dict(torch.load("best_attention_lstm.pt"))
model.eval()

misclassified = []
correct = []

with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
        outputs = model(batch_X)
        preds = (outputs.cpu().numpy() > 0.5).astype(int)
        labels = batch_y.numpy().astype(int)

        for i in range(len(preds)):
            if preds[i] != labels[i]:
                misclassified.append((preds[i], labels[i], batch_X[i].cpu().numpy()))
            else:
                correct.append((preds[i], labels[i], batch_X[i].cpu().numpy()))

# Load tokenizer index to word map
index_word = {v: k for k, v in tokenizer.word_index.items()}

# Decode sequence

def decode_sequence(sequence):
    return " ".join([index_word.get(i, "") for i in sequence if i != 0])

print("\n❌ Misclassified Examples (Prediction vs Actual):")
for i, (pred, true, seq) in enumerate(misclassified[:5]):
    print(f"\nExample {i+1}:")
    print(f"Predicted: {'Recommended' if pred == 1 else 'Not Recommended'}")
    print(f"Actual: {'Recommended' if true == 1 else 'Not Recommended'}")
    print(f"Review (decoded): {decode_sequence(seq)}")

print("\nCorrectly Classified Examples:")
for i, (pred, true, seq) in enumerate(correct[:3]):
```

```
    print(f"\nExample {i+1}:")
    print(f"Correct Label: {'Recommended' if true == 1 else 'Not Recommended'}")
    print(f"Review (decoded): {decode_sequence(seq)}")

# Discussion Note (for report):
# - Misclassifications often happen with reviews that include sarcasm, overly neutral or ambiguous wording.
# - Very short reviews or those lacking sentiment words may also lead to errors.
# - Long reviews with mixed sentiment ("The flight was late but the crew was amazing") can confuse the classifier.
```

```
Misclassified Examples (Prediction vs Actual):

Example 1:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): leeds bradford la vega via heathrow customer service handling question never answered three time asked explain cost one checked bag caused fare increase £ ou

Example 2:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): £ decided take website wouldnt allow choose seat check phoned ba happy call picked quickly told changed booking check airport turned three hour flight told t

Example 3:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): wonderful service flight edinburghflorence july mainline british airway could learn lot ba cityflyer franchise partner africa com air

Example 4:
Predicted: Recommended
Actual: Not Recommended
Review (decoded): london heathrow lisbon return british airway decent aircraft service onboard dreadful pilot first officer sounded friendly professional kept u date flight de

Example 5:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): flight slight trepidation disappointed though flight smooth left time seating fairly comfortable consider minute flight preboard board announcement efficient

Correctly Classified Examples:

Example 1:
Correct Label: Not Recommended
Review (decoded): <OOV> disappointed outbound flight food choice gone reached th row economy cabin meaning rest cabin second choice meal second choice good reason truly unplea

Example 2:
Correct Label: Not Recommended
Review (decoded): arrangement whereby face neighbour hot towel given like warm rag comparison airline food ok selection menu however service time merely passed tray neighbour

Example 3:
Correct Label: Recommended
Review (decoded): flew british airway oslo london heathrow march crew happy helpful cabin clean looked new wasnt new plane seat new served turkey cheese croissant fly ba time
```

Stacked LSTM This step identifies prediction errors made by the Stacked LSTM model by decoding and reviewing misclassified inputs. Errors typically arise from ambiguous, mixed-sentiment, or context-limited reviews. These insights help guide improvements like extending input length or using richer embeddings.

```
# Error Analysis for Stacked LSTM Model (Stacked LSTM)

# Load best performing stacked LSTM model
model = StackedLSTM(vocab_size=MAX_VOCAB_SIZE, embedding_dim=128, hidden_dim=64)
model.load_state_dict(torch.load("best_stacked_lstm.pt"))
model.to(device)
model.eval()

misclassified = []
correct = []

with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
        outputs = model(batch_X)
        preds = (outputs.cpu().numpy() > 0.5).astype(int)
        labels = batch_y.numpy().astype(int)

        for i in range(len(preds)):
            if preds[i] != labels[i]:
                misclassified.append((preds[i], labels[i], batch_X[i].cpu().numpy()))
            else:
                correct.append((preds[i], labels[i], batch_X[i].cpu().numpy()))

# Load tokenizer index to word map
index_word = {v: k for k, v in tokenizer.word_index.items()}

# Decode sequence

def decode_sequence(sequence):
    return " ".join([index_word.get(i, "") for i in sequence if i != 0])

print("\n❌Misclassified Examples (Stacked LSTM: Prediction vs Actual):")
for i, (pred, true, seq) in enumerate(misclassified[:5]):
    print(f"\nExample {i+1}:")
    print(f"Predicted: {'Recommended' if pred == 1 else 'Not Recommended'}")
    print(f"Actual: {'Recommended' if true == 1 else 'Not Recommended'}")
    print(f"Review (decoded): {decode_sequence(seq)}")

print("\nCorrectly Classified Examples (Stacked LSTM):")
for i, (pred, true, seq) in enumerate(correct[:3]):
    print(f"\nExample {i+1}:")
    print(f"Correct Label: {'Recommended' if true == 1 else 'Not Recommended'}")
    print(f"Review (decoded): {decode_sequence(seq)}")

# Note for report:
# Common errors may include:
# - Mixed or contradictory sentiment
# - Ambiguous or neutral phrasing
# - Missing context due to fixed-length input truncation
# Suggestions:
```

```
# - Consider using longer max_seq_len
# - Experiment with pretrained embeddings
# - Test transformer models for richer contextual understanding
```

```
❌ Misclassified Examples (Stacked LSTM: Prediction vs Actual):

Example 1:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): leeds bradford la vega via heathrow customer service handling question never answered three time asked explain cost one checked bag caused fare increase £ ou

Example 2:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): mumbai boston via london flight british airway really good catering excellent flight even service good however seat limited legroom due ife box blocking spac

Example 3:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): £ decided take website wouldnt allow choose seat check phoned ba happy call picked quickly told changed booking check airport turned three hour flight told t

Example 4:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): flew edinburgh new york via lhr th june flight full service check board good complaint breakfast return flight croissant jam tea coffee disappointing

Example 5:
Predicted: Recommended
Actual: Not Recommended
Review (decoded): prague denver via london almost missed flight checkin complete really important paper took minute later one asked paper flight heathrow average expect hour f

✅ Correctly Classified Examples (Stacked LSTM):

Example 1:
Correct Label: Not Recommended
Review (decoded): <OOV> disappointed outbound flight food choice gone reached th row economy cabin meaning rest cabin second choice meal second choice good reason truly unplea

Example 2:
Correct Label: Not Recommended
Review (decoded): arrangement whereby face neighbour hot towel given like warm rag comparison airline food ok selection menu however service time merely passed tray neighbour

Example 3:
Correct Label: Recommended
Review (decoded): flew british airway oslo london heathrow march crew happy helpful cabin clean looked new wasnt new plane seat new served turkey cheese croissant fly ba time
```

Bidirectional LSTM This analysis highlights misclassified reviews to understand the model's limitations. Errors often stem from mixed opinions or subtle sentiments not easily captured by standard LSTMs. Enhancing context handling using attention or transformer-based architectures can improve performance.

```
# Error Analysis for Bidirectional LSTM Model

# Load best performing bidirectional LSTM model
model = LSTMClassifier(vocab_size=MAX_VOCAB_SIZE, embedding_dim=128, hidden_dim=64)
model.load_state_dict(torch.load("best_lstm_model.pt"))
model.to(device)
model.eval()

misclassified = []
correct = []

with torch.no_grad():
    for batch_X, batch_y in test_loader:
        batch_X = batch_X.to(device)
        outputs = model(batch_X)
        preds = (outputs.cpu().numpy() > 0.5).astype(int)
        labels = batch_y.numpy().astype(int)

        for i in range(len(preds)):
            if preds[i] != labels[i]:
                misclassified.append((preds[i], labels[i], batch_X[i].cpu().numpy()))
            else:
                correct.append((preds[i], labels[i], batch_X[i].cpu().numpy()))

# Load tokenizer index to word map
index_word = {v: k for k, v in tokenizer.word_index.items()}

# Decode sequence

def decode_sequence(sequence):
    return " ".join([index_word.get(i, "") for i in sequence if i != 0])

print("\n❌ Misclassified Examples (Bidirectional LSTM: Prediction vs Actual):")
for i, (pred, true, seq) in enumerate(misclassified[:5]):
    print(f"\nExample {i+1}:")
    print(f"Predicted: {'Recommended' if pred == 1 else 'Not Recommended'}")
    print(f"Actual: {'Recommended' if true == 1 else 'Not Recommended'}")
    print(f"Review (decoded): {decode_sequence(seq)}")

print("\nCorrectly Classified Examples (Bidirectional LSTM):")
for i, (pred, true, seq) in enumerate(correct[:3]):
    print(f"\nExample {i+1}:")
    print(f"Correct Label: {'Recommended' if true == 1 else 'Not Recommended'}")
    print(f"Review (decoded): {decode_sequence(seq)}")

# Note for report:
# - Errors may be due to mixed or neutral language.
# - Some misclassifications occur in reviews with contrasting phrases (e.g., "bad food but good service").
# - Improvements could include attention mechanisms or transformers to better model long-range dependencies.
```

```
❌ Misclassified Examples (Bidirectional LSTM: Prediction vs Actual):

Example 1:
Predicted: Recommended
Actual: Not Recommended
```

```
Review (decoded): deeply unimpressed lack social distancing flight plane london full social distancing could applied number people wearing face mask incorrectly mouth chinstra

Example 2:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): leeds bradford la vega via heathrow customer service handling question never answered three time asked explain cost one checked bag caused fare increase £ ou

Example 3:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): mumbai boston via london flight british airway really good catering excellent flight even service good however seat limited legroom due ife box blocking spac

Example 4:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): £ decided take website wouldnt allow choose seat check phoned ba happy call picked quickly told changed booking check airport turned three hour flight told t

Example 5:
Predicted: Not Recommended
Actual: Recommended
Review (decoded): chicago london heathrow terrific inflight service food updated inflight entertainment service great large selection film tv show menu extensive nicely presen

✅ Correctly Classified Examples (Bidirectional LSTM):

Example 1:
Correct Label: Not Recommended
Review (decoded): <OOV> disappointed outbound flight food choice gone reached th row economy cabin meaning rest cabin second choice meal second choice good reason truly unplea

Example 2:
Correct Label: Not Recommended
Review (decoded): arrangement whereby face neighbour hot towel given like warm rag comparison airline food ok selection menu however service time merely passed tray neighbour

Example 3:
Correct Label: Recommended
Review (decoded): flew british airway oslo london heathrow march crew happy helpful cabin clean looked new wasnt new plane seat new served turkey cheese croissant fly ba time
```

Step 7: Visualization of Results In this step, key performance metrics such as accuracy, confusion matrix, and classification report are visualized to interpret model behavior. These visual tools help identify class-wise performance, highlight misclassification patterns, and support model comparison. Visualization reinforces understanding of model strengths and weaknesses—an essential practice in NLP coursework evaluation.

LSTM Model Accuracy Comparison Visualization of model performance helps interpret and compare different LSTM architectures effectively. The accuracy comparison bar chart clearly illustrates how advanced models like Stacked LSTM and Attention LSTM outperform the baseline. Such visual tools aid in making informed decisions on model selection and future improvements.

```python
import matplotlib.pyplot as plt

# Model names and accuracies
models = [
    "Baseline LSTM",
    "Bidirectional LSTM",
    "Stacked LSTM",
    "Attention LSTM"
]
accuracies = [0.7665, 0.8381, 0.8556, 0.8394]

# Create the plot
plt.figure(figsize=(10, 6))
bars = plt.bar(models, accuracies, color=['skyblue', 'mediumseagreen', 'gold', 'lightcoral'])
plt.ylim(0.7, 0.9)
plt.ylabel("Accuracy")
plt.title("LSTM Model Accuracy Comparison")

# Add accuracy labels on top of bars
for bar, acc in zip(bars, accuracies):
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.005, f"{acc:.4f}", ha='center', va='bottom', fontsize=10)

plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```