Coursework Assignment: Comparative text classification using statistical and embedding-based models

Introduction

Domain-specific area: Mental Health Sentiment Analysis

In recent years, the field of mental health has gained significant attention within public health and societal discourse. Issues such as anxiety, depression, and stress have become increasingly prevalent, particularly exacerbated by global challenges such as the COVID-19 pandemic. This has underscored the urgent need for effective tools to monitor and comprehend emotional well-being. One promising approach involves analyzing textual data sourced from social media platforms, online forums, and mental health surveys to discern public sentiment and identify patterns of emotion.

Mental health sentiment analysis entails categorizing textual information based on expressed emotions such as joy, sadness, anger, fear, surprise, and love. This analytical process holds potential benefits for mental health professionals, researchers, and policymakers by offering insights into the emotional landscapes of populations, identifying emerging trends in mental health, and facilitating timely interventions. The significance of this area is substantiated by numerous studies demonstrating how sentiment analysis can detect mental health conditions and provide insights into public mood and well-being (Calvo et al., 2017; Guntuku et al., 2019).

Given the critical nature of mental health and the abundance of textual data available for analysis, this domain presents a valuable opportunity to evaluate the efficacy of both traditional statistical models and modern embedding-based models in text classification tasks. Research findings in this area could significantly contribute to the broader field of mental health informatics, enhancing the development of more precise and efficient tools for sentiment analysis.

Objectives

The main objective of this coursework is to explore and compare the effectiveness and suitability of statistical models and embedding-based models in text classification tasks, specifically within a domain on mental health sentiment analysis. The specific goals of this exploration are as follows:

1. **Evaluate the Performance of Statistical Models**
    - Evaluate the performance of traditional statistical models, such as Naive Bayes and Support Vector Machines (SVM), in classifying emotions in text data.
    - Measure the accuracy, precision, recall, and F1-score of these models in predicting different emotional states.

- Identify the advantages and limitations of statistical models in dealing with varied and complex text data related to mental health.

2. **Examine the Effectiveness of Embedding-Based Models**
    - Examine the effectiveness of modern embedding-based models, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), in the same classification tasks.
    - Compare the results with those of statistical models using the same evaluation metrics.
    - Determine how well embedding-based models capture contextual nuances and semantic meanings in text, which are crucial for accurate emotion detection.

3. **Compare and Contrast Both Approaches**
    - Conduct a detailed comparison between statistical and embedding-based models to understand their relative strengths and weaknesses.
    - Evaluate the computational efficiency, scalability, and practicality of implementing these models in real-world mental health monitoring systems.

4. **Address Text Classification Challenges**:
    - Identify and tackle specific challenges related to text classification in mental health sentiment analysis, such as managing noisy and context-dependent language, handling imbalanced datasets, and ensuring robustness across diverse textual sources.

5. **Contribute to Mental Health Informatics**
    - Provide insights into how these models can improve the accuracy and reliability of sentiment analysis tools used by mental health professionals.
    - Highlight the potential of advanced NLP techniques in enhancing mental health interventions and policymaking.
    - Contribute to the existing literature by presenting empirical evidence on the performance of different text classification models in the context of mental health sentiment analysis.

The outcomes of this study offer valuable guidance to researchers, practitioners, and policymakers involved in leveraging computational approaches for mental health analysis. By building upon existing research, this study aims to enhance the accuracy and reliability of sentiment analysis tools, thereby supporting more informed decision-making and interventions in mental health.

Dataset Description

The dataset used in this study is the "Emotions Dataset for NLP" which is sourced from Kaggle, a platform widely used for sharing and acessing datasets for machine learning and data science projects. This dataset is designed specifically for NLP tasks involved emotion classification.

The dataset contains a total of about 20,000 labeled instances of textual data and is divided into three subsets: training set, validation set and test set. The training set comprises 16,000 instances, the validation set contains 2,000 instances and the test set includes 2,000 instances. Each instance is a short text snippet, typically a sentence or a short paragraph, labeled with one of the six corresponding emotions: joy, sadness, anger, fear, surprise and disgust.

The dataset was curated from a variety of sources, including social media posts, online forums, and other publicly available textual content. The data collection process invloved extracting short text snippets and manually labeling them with the appropriate emotion categories. Also, the labeling process was likely conducted by human annotators who classified the emotions based on the context and content of each text snippet.

The diverse range of text samples labeled with distinct emotions provided by the dataset helps to facilitate the comparison of different text classification models. Furthermore, the balanced distribution of emotions across the dataset ensures that each model's performance can be evaluated comprehensively across various emotional categories. Its structured format and clear labeling make it well-suited for training and testing both statistical and embedding-based models in the realm of mental health sentiment analysis.

Evaluation Methodology

Several standard evaluation metrics will be used to assess the performance of the statistical and embedding-based models in the task of emotion classification. These metrics provide a comprehensive understanding of each model's effectiveness and allow for a direct comaprison between the two methodologies. These metrics are:

1. **Accuracy**: Measures the ratio of correctly classified instances to the total instances. It is calculated as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of predictions}$$

   Accuracy offers an overall indication of model performance across all classes.

1. **Precision**: Measures the proportion of true positive predictions out of all positive predictions made by the model. It is defined as:

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)}$$

   Precision is useful when minimizing false positives is crucial.

1. **Recall (Sensitivity)**: Measures the proportion of true positive predictions out of all actual positive instances. It is given by:

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

   Recall is important in scenarios where capturing all positive instances is essential.

1. **F1-Score**: The F1-Score is the harmonic mean of precision and recall, providing a balance between the two. It is calculated as:

$$F1 = 2 * \frac{Precision(P) * Recall(R)}{Precision(P) + Recall(R)}$$

This metric is particularly useful for imbalanced datasets, as it accounts for both false positives and false negatives.

1. **Confusion Matrix**: A table that details the performance of the model by showing the true positives, false positives, true negatives, and false negatives for each class. It aids in understanding specific areas of misclassification and overall model performance.

To compare the performance of statistical models with embedding-based models, the following steps will be taken:

1. **Model Training and Validation**: Train each model on the training set and fine-tune their hyperparameters using the validation set.
2. **Performance Evaluation**: Evaluate each model on the test set using the specified metrics. Record the accuracy, precision, recall, F1-score, and confusion matrix for each model.
3. **Comparative Analysis**: Compare the results of statistical models with those of embedding-based models across all metrics. This comparison will highlight the strengths and weaknesses of each approach in emotion classification tasks.
4. **Statistical Significance**: Conduct statistical tests, such as paired t-tests, to determine if the differences in performance metrics between the two methodologies are statistically significant.

By systematically applying these evaluation metrics, we can thoroughly compare the performance of statistical and embedding-based models, providing deeper insights into their effectiveness for classifying emotions in mental health-related textual data.

Implementation

Data Preprocessing

In this step, the dataset is locally stored and preprocessed for model training. Since the dataset consists of text document files, it is necesssary to read and process these files to extract the text and corresponding labels.

In [1]:
```
# Import the necessary libraries
import nltk
import string
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report

# Directory path for my datasets
test_dir = r"G:\UOL BSc Comp Sci\Y3S1\NLP\Mid-Term\emotions dataset_kaggle\test.txt
train_dir = r"G:\UOL BSc Comp Sci\Y3S1\NLP\Mid-Term\emotions dataset_kaggle\train.t
validation_dir = r"G:\UOL BSc Comp Sci\Y3S1\NLP\Mid-Term\emotions dataset_kaggle\va

# Function to load data from a directory
def load_data(file_path):
    texts = []
    labels = []
    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            text, label = line.strip().split(';')
            texts.append(text)
            labels.append(label)
    return texts, labels
```

In order for the dataset to be tailored to both statistical and embedding-based models, some preprocessing steps must be done to the text document files.

The steps needed for my text are:

- Tokenization: Split text into individual words or tokens
- Text Cleaning: Remove unnecessary characters (e.g. punctuations) and convert text to lowercase to ensure uniformity
- Stopwords Removal: Remove common words (e.g. 'the', 'and', 'is' etc.) that do not contribute significantly to the meaning of the text
- Lemmatization: Reduce words to their base or root form (e.g. 'running' to 'run')

Although stemming was considered in the preprocessing, I still opted for lemmatization as the context of texts plays a huge role in maintaining the core meaning and sentiment.

In [2]:
```
# Preprocess the texts with lemmatization, lowercasing and stopword removal
def preprocess_texts(texts):
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    processed_texts = []
    for text in texts:
        # Tokenization
        tokens = word_tokenize(text)
        # Lowercasing
        tokens = [word.lower() for word in tokens]
        # Removing punctuation and stopwords
        tokens = [word for word in tokens if word not in stop_words and word not ir
        # Lemmatization
        tokens = [lemmatizer.lemmatize(word) for word in tokens]
        processed_texts.append(tokens)

    return processed_texts


# Load datasets
test_texts, test_labels= load_data(test_dir)
```

```
train_texts, train_labels = load_data(train_dir)
validation_texts, validation_labels = load_data(validation_dir)

# Convert to DataFrames
test_df = pd.DataFrame({'text': test_texts, 'label': test_labels})
train_df = pd.DataFrame({'text': train_texts, 'label': train_labels})
validation_df = pd.DataFrame({'text': validation_texts, 'label': validation_labels})

print("Test DataFrame")
print(test_df.head())
print('\n')
print(test_df.info())

print("\nTraining DataFrame")
print(train_df.head())
print('\n')
print(train_df.info())

print("\nValidation DataFrame")
print(validation_df.head())
print('\n')
print(validation_df.info())
```

```
Test DataFrame
                                                text    label
0  im feeling rather rotten so im not very ambiti...  sadness
1          im updating my blog because i feel shitty  sadness
2  i never make her separate from me because i do...  sadness
3  i left with my bouquet of red and yellow tulip...      joy
4    i was feeling a little vain when i did this one  sadness


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    2000 non-null   object
 1   label   2000 non-null   object
dtypes: object(2)
memory usage: 31.4+ KB
None

Training DataFrame
                                                text    label
0                           i didnt feel humiliated  sadness
1  i can go from feeling so hopeless to so damned...  sadness
2   im grabbing a minute to post i feel greedy wrong    anger
3  i am ever feeling nostalgic about the fireplac...     love
4                              i am feeling grouchy    anger


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16000 entries, 0 to 15999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    16000 non-null  object
 1   label   16000 non-null  object
dtypes: object(2)
memory usage: 250.1+ KB
None

Validation DataFrame
                                                text    label
0  im feeling quite sad and sorry for myself but ...  sadness
1  i feel like i am still looking at a blank canv...  sadness
2                     i feel like a faithful servant     love
3                 i am just feeling cranky and blue    anger
4  i can have for a treat or if i am feeling festive      joy


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    2000 non-null   object
 1   label   2000 non-null   object
dtypes: object(2)
memory usage: 31.4+ KB
None
```

Text Representation

Based on the three dataframes, I will be converting text data into numerical features using the Bag of Words (BoW) representation. In BoW, a document is reprsented as a multiset (bag) of its words, disregarding grammar and word order but keeping multiplicity.

The BoW representation was chosen due to its simplicity, interpretability and effectiveness in handling the specific characteristics of textual data related to emotional sentiment analysis. As it scales well with the size of the datset and the vocabulary, it can handle large corpora efficiently, making it feasible to process extensive datasets such as the one used in this project with thousands of labeled instances. BoW also serves as a foundational approach that can be compared against more advanced techniques like word embeddings.

The model involves creating a frequency table or term-document matrix where each row represents a document and each column represents a term. The entries in the matrix are the frequencies of the terms in the documents.

In [3]:
```
train_tokens = preprocess_texts(train_texts)
validation_tokens = preprocess_texts(validation_texts)
test_tokens = preprocess_texts(test_texts)

# Convert tokens to text strings for BoW representation
train_texts_bow = [' '.join(tokens) for tokens in train_tokens]
validation_texts_bow = [' '.join(tokens) for tokens in validation_tokens]
test_texts_bow = [' '.join(tokens) for tokens in test_tokens]

# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit and transform on training data
X_train_bow = vectorizer.fit_transform(train_texts_bow)

# Transform validation and test data
X_val_bow = vectorizer.transform(validation_texts_bow)
X_test_bow = vectorizer.transform(test_texts_bow)

# Optionally, convert labels to numeric form if necessary
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
validation_labels_encoded = label_encoder.transform(validation_labels)
test_labels_encoded = label_encoder.transform(test_labels)
```

Differences in Data Preparation

The preprocessing steps for both statistical and embedding-based models differ significantly due to the distinct ways the models represent and utilize textual data.

Statistical Models

For statistical models such as Naive Bayes and Support Vector Machines (SVM), the focus is on frequency-based representations of text. These models typically use simple, yet effective, techniques like Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). They rely on these frequency-based representations because they are straightforward and computationally efficient. Their primary goal is to convert the text into numerical features that these models can process.

Embedding-Based Models

Embedding-based models, such as Long Short-Term Memory (LSTM) networks and Bidirectional Encoder Representations from Transformers (BERT), use dense vector representations of text that capture the semantic meaning and relationships between words. They benefit from these dense vector representations as they capture richer information about the words and their contexts, enabling the models to understand and classify text more effectively.

Baseline Performance

For the Baseline Performance model, a Naive Bayes classifier is used with the co-existing BoW text representation created during the data preprocessing section.

The classifier was chosen due to two reasons:

- It is one of the more straightforward models to implement and requires minimal computational resources
- Provides clear insights into the decision-making process when its predictions are based on the probabilities of words given by the class labels

The steps involved in achieving this are as follows:

- Train the classifier using scikit-learn
- Make predictions of the baseline model
- Evaluate the performance using standard metrics

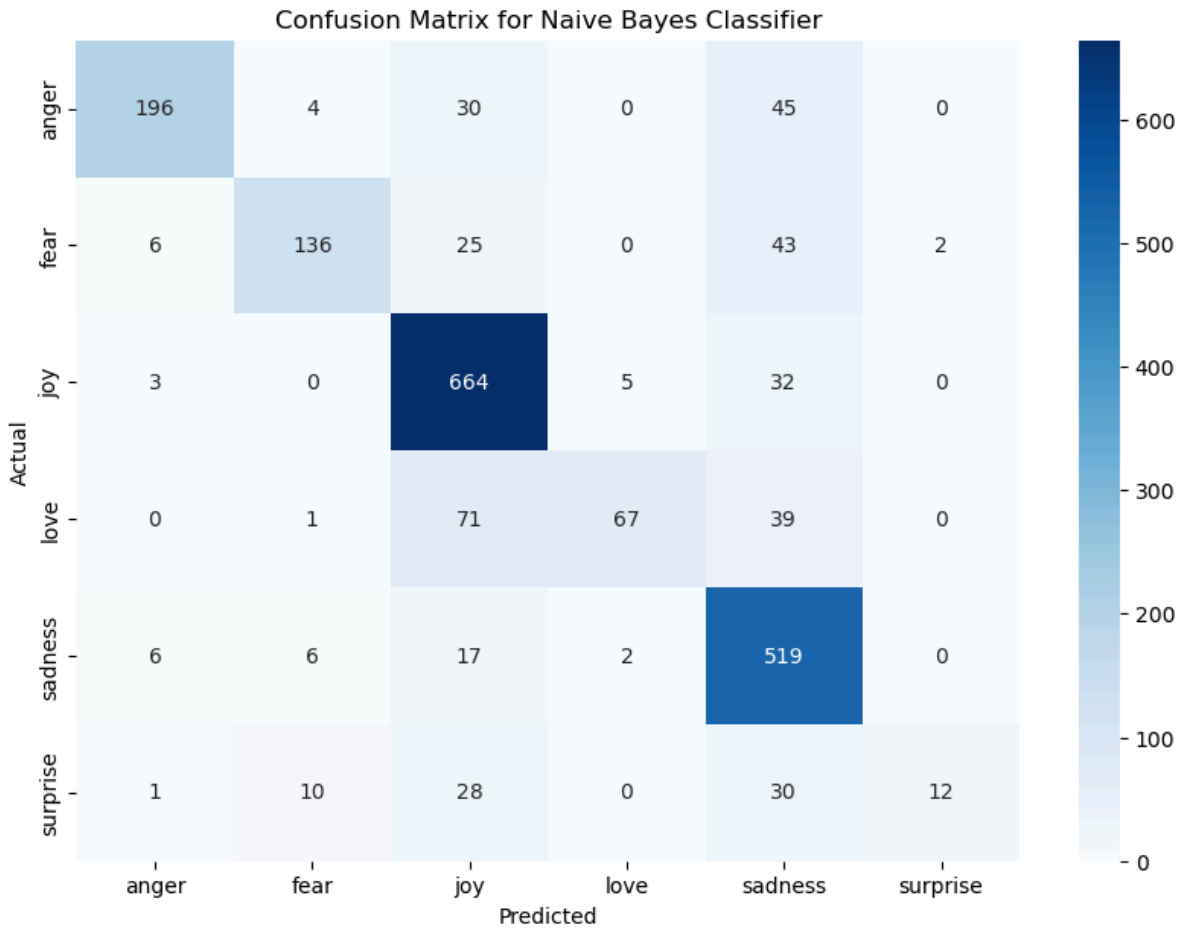
In [4]:
```
# Import sklearn modules for use of Naive Bayes and its relevant features
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Train Naive Bayes Classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_bow, train_labels_encoded)

# Predict on validation set
val_predictions = nb_classifier.predict(X_val_bow)

# Evaluate performance
accuracy = accuracy_score(validation_labels_encoded, val_predictions)
precision = precision_score(validation_labels_encoded, val_predictions, average='we
recall = recall_score(validation_labels_encoded, val_predictions, average='weighted
f1 = f1_score(validation_labels_encoded, val_predictions, average='weighted')

print(f"Baseline Naive Bayes Performance:\nAccuracy: {accuracy}\nPrecision: {precis
```

```
Baseline Naive Bayes Performance:
Accuracy: 0.797
Precision: 0.8157424198616934
Recall: 0.797
F1-score: 0.7770179763088395
```

A confusion matrix is also added to visualize the performance of the Naive Bayes model.

In [5]:
```
# Calculate the confusion matrix
conf_matrix = confusion_matrix(validation_labels_encoded, val_predictions)
```

```
# Define the class names
class_names = label_encoder.classes_

# Create a DataFrame for the confusion matrix
conf_matrix_df = pd.DataFrame(conf_matrix, index=class_names, columns=class_names)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Naive Bayes Classifier')
plt.show()
```



Comparative Classification Approach

In this section, two distinct approaches will be implemented and compared for text classification purposes: a traditional statistical model and a modern deep learning model. For the statistical model approach, a Support Vector Machine (SVM) with a linear kernel will be used. For the deep learning approach, a Bidirectional Long Short-Term Memory (BiLSTM) network will be implemented.

Traditional Statistical Model: Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a powerful classifier that works by finding the hyperplane that best separates the data into different classes. For text classification, the purpose of a linear kernel being used is due to its simplicity and effectiveness in high-dimensional spaces typical of text data represented as Bag of Words (BoW).

In [6]:
```
# Import sklearn modules for use of SVM and its relevant features
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Initialize the SVM classifier with a linear kernel
svm_classifier = SVC(kernel='linear')

# Train the classifier
svm_classifier.fit(X_train_bow, train_labels_encoded)

# Predict on validation set
svm_val_predictions = svm_classifier.predict(X_val_bow)

# Evaluate performance
svm_accuracy = accuracy_score(validation_labels_encoded, svm_val_predictions)
svm_precision = precision_score(validation_labels_encoded, svm_val_predictions, ave
svm_recall = recall_score(validation_labels_encoded, svm_val_predictions, average='
svm_f1 = f1_score(validation_labels_encoded, svm_val_predictions, average='weighted

print(f"SVM Performance:\nAccuracy: {svm_accuracy}\nPrecision: {svm_precision}\nRec
```

```
SVM Performance:
Accuracy: 0.8885
Precision: 0.8890984990762583
Recall: 0.8885
F1-score: 0.888723241053716
```

Strengths & Weaknesses

- Strengths:
  - **Effective in High-Dimensional Spaces**: The SVM's ability to handle high dimensional data is beneficial when dealing with text data converted into BoW representations, which often results in a large feature space.
  - **Memory Efficient**: It uses a subset of training points (support vectors) to make decisions, which is helpful when working with large datasets like our emotions dataset.
  - **Clear Interpretability**: The SVM provides clear interpretability through feature weights, aiding in understanding which words contribute most to each emotion classification.

- Weaknesses:
  - **Sensitivity to Hyperparameters**: The performance of SVM can be highly dependent on the choice of hyperparameters, requiring extensive tuning.
  - **Handling Noisy Data**: SVMs can struggle with noisy text data, which is common in social media and survey responses related to mental health.
  - **Context Ignorance**: The SVM's BoW approach does not capture the context and semantics of words, which is critical in understanding the nuances of emotional expression in text.

Modern Deep Learning Model: Bidirectional Long Short-Term Memory (BiLSTM)

A Bidirectional Long Short-Term Memory (BiLSTM) network is an advanced neural network model that can learn from sequential data in both forward and backward directions. This

architecture is particularly useful for capturing context in text data, which is crucial for sentiment analysis.

In order to train embedding-based models like BiLSTM, the textual data given in the dataset should be converted into word embeddings using the 'tensorflow' library. Adam optimizer and catergorial cross-entropy loss are also implemented to further optimize the model, before finally evaluating the model on the validation set.

In [7]:
```
# Import tensorflow modules for use of BiLSTM and its relevant features
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense
from tensorflow.keras.optimizers import Adam

# Tokenize and pad sequences
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(train_texts)
X_train_seq = tokenizer.texts_to_sequences(train_texts)
X_val_seq = tokenizer.texts_to_sequences(validation_texts)
X_test_seq = tokenizer.texts_to_sequences(test_texts)

max_len = 100
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_val_pad = pad_sequences(X_val_seq, maxlen=max_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# Build BiLSTM model
model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_len))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(6, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',

# Convert labels to categorical
train_labels_categorical = tf.keras.utils.to_categorical(train_labels_encoded, num_
val_labels_categorical = tf.keras.utils.to_categorical(validation_labels_encoded, r

# Train the model
model.fit(X_train_pad, train_labels_categorical, epochs=10, batch_size=64, validati

# Evaluate performance
val_loss, val_accuracy = model.evaluate(X_val_pad, val_labels_categorical)
print(f"BiLSTM Performance:\nAccuracy: {val_accuracy}")
```

```
Epoch 1/10
```

```
C:\Users\Marcus Tan\anaconda3\lib\site-packages\keras\src\layers\core\embedding.p
y:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 9s 29ms/step - accuracy: 0.4068 - loss: 1.4787 - val_
accuracy: 0.7580 - val_loss: 0.7368
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 27ms/step - accuracy: 0.8407 - loss: 0.4965 - val_
accuracy: 0.9030 - val_loss: 0.3058
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 28ms/step - accuracy: 0.9445 - loss: 0.1811 - val_
accuracy: 0.9105 - val_loss: 0.2993
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 28ms/step - accuracy: 0.9682 - loss: 0.0973 - val_
accuracy: 0.9165 - val_loss: 0.2498
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 27ms/step - accuracy: 0.9816 - loss: 0.0566 - val_
accuracy: 0.9200 - val_loss: 0.2346
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 28ms/step - accuracy: 0.9876 - loss: 0.0402 - val_
accuracy: 0.9185 - val_loss: 0.2647
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 28ms/step - accuracy: 0.9893 - loss: 0.0330 - val_
accuracy: 0.9145 - val_loss: 0.2694
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 28ms/step - accuracy: 0.9896 - loss: 0.0307 - val_
accuracy: 0.9205 - val_loss: 0.2774
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 28ms/step - accuracy: 0.9859 - loss: 0.0489 - val_
accuracy: 0.9180 - val_loss: 0.2820
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 7s 27ms/step - accuracy: 0.9913 - loss: 0.0249 - val_
accuracy: 0.9195 - val_loss: 0.3041
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9267 - loss: 0.2624
BiLSTM Performance:
Accuracy: 0.9194999933242798
```

In [8]:

```python
# Predict on validation set using BiLSTM
bilstm_val_predictions = model.predict(X_val_pad)
bilstm_val_predictions = bilstm_val_predictions.argmax(axis=1)

# Calculate BiLSTM metrics
bilstm_accuracy = accuracy_score(validation_labels_encoded, bilstm_val_predictions)
bilstm_precision = precision_score(validation_labels_encoded, bilstm_val_prediction
bilstm_recall = recall_score(validation_labels_encoded, bilstm_val_predictions, ave
bilstm_f1 = f1_score(validation_labels_encoded, bilstm_val_predictions, average='we

print(f"BiLSTM Performance:\nAccuracy: {bilstm_accuracy}\nPrecision: {bilstm_precis
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
BiLSTM Performance:
Accuracy: 0.9195
Precision: 0.9196746926239521
Recall: 0.9195
F1-score: 0.9193719036050388
```

Strengths and Weaknesses

- Strengths:
  - **Captures Long-Term Dependencies**: The BiLSTM's ability to capture long-term dependencies is crucial for understanding the context of emotions in text, which often relies on the sequence of words.
  - **Bidirectional Learning**: Learning from both forward and backward contexts helps in better understanding the sentiment expressed in complex sentences, which is particularly valuable in nuanced mental health texts.

    - **Handles Noisy Data**: BiLSTMs are more robust to noisy data due to their ability to learn from context and sequence information, which is common in user-generated content related to mental health.

- Weaknesses:
    - **Computationally Intensive**: Training BiLSTM models requires significant computational resources and time, especially with large datasets.
    - **Data Hungry**: BiLSTMs perform best with large amounts of data, which can be a limitation if annotated mental health data is scarce.
    - **Complexity and Interpretability**: The complexity of BiLSTM models can make them difficult to interpret, which is a drawback when trying to understand the features influencing emotion classification.

Conclusions

Performance Analysis & Comparative Discussion

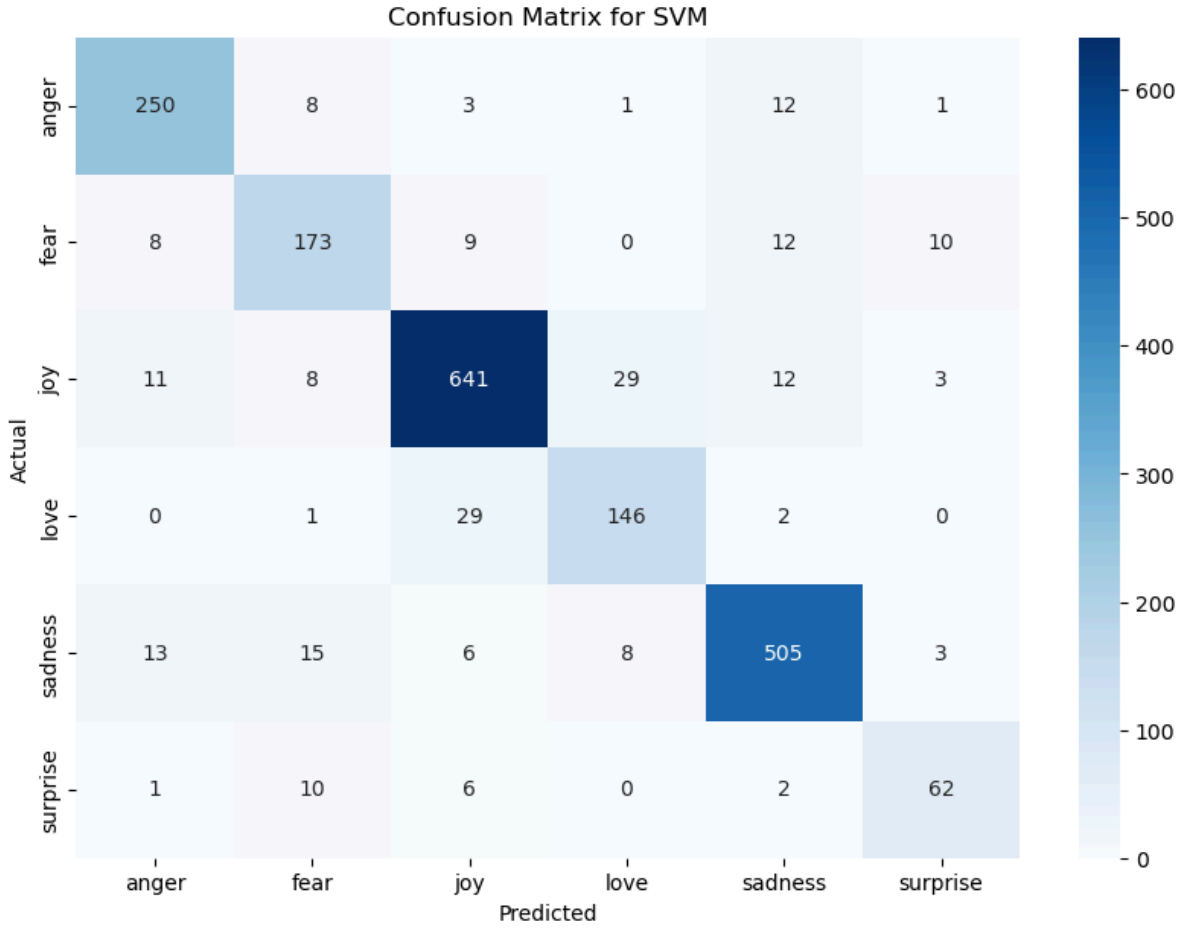
The key performance metrics for both models (e.g. accuracy, precision, recall and F1-score) are summarized into a classification report.

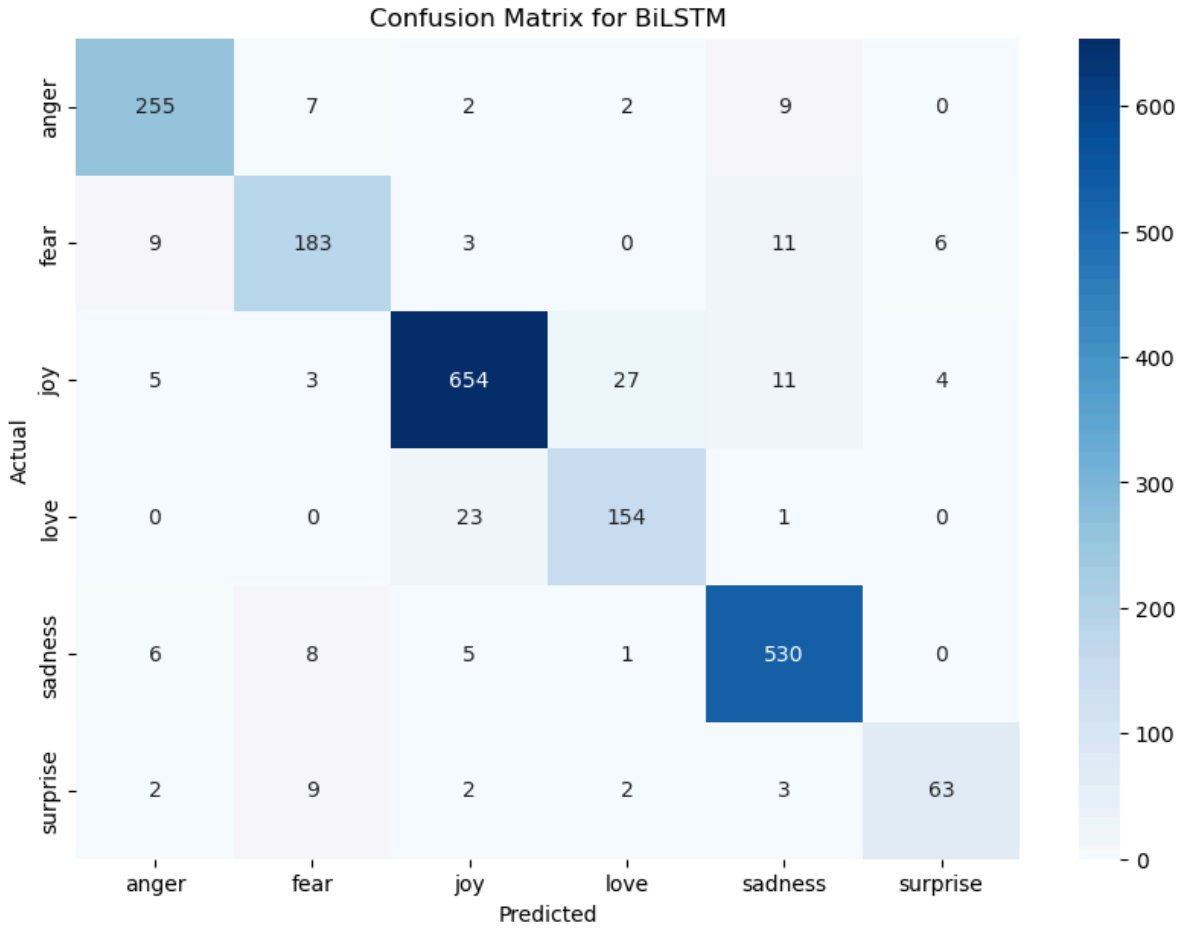
In [9]:
```
# SVM Metrics
svm_report = classification_report(validation_labels_encoded, svm_val_predictions,
print("SVM Classification Report:\n", svm_report)

# BiLSTM Metrics
bilstm_report = classification_report(validation_labels_encoded, bilstm_val_predict
print("BiLSTM Classification Report:\n", bilstm_report)
```

```
SVM Classification Report:
               precision    recall  f1-score   support

       anger       0.88      0.91      0.90       275
        fear       0.80      0.82      0.81       212
         joy       0.92      0.91      0.92       704
        love       0.79      0.82      0.81       178
     sadness       0.93      0.92      0.92       550
    surprise       0.78      0.77      0.77        81

    accuracy                           0.89      2000
   macro avg       0.85      0.86      0.85      2000
weighted avg       0.89      0.89      0.89      2000

BiLSTM Classification Report:
               precision    recall  f1-score   support

       anger       0.92      0.93      0.92       275
        fear       0.87      0.86      0.87       212
         joy       0.95      0.93      0.94       704
        love       0.83      0.87      0.85       178
     sadness       0.94      0.96      0.95       550
    surprise       0.86      0.78      0.82        81

    accuracy                           0.92      2000
   macro avg       0.90      0.89      0.89      2000
weighted avg       0.92      0.92      0.92      2000
```

For better visualization of the performance of both models, confusion matrices are used to provide insights into the classification accuracy for each emotion class.

In [10]:
```
# Confusion Matrix for SVM
svm_conf_matrix = confusion_matrix(validation_labels_encoded, svm_val_predictions)
plt.figure(figsize=(10, 7))
sns.heatmap(svm_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_e
plt.title('Confusion Matrix for SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Confusion Matriox for BiLSTM
bilstm_conf_matrix = confusion_matrix(validation_labels_encoded, bilstm_val_predict
plt.figure(figsize=(10, 7))
sns.heatmap(bilstm_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labe
plt.title('Confusion Matrix for BiLSTM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```





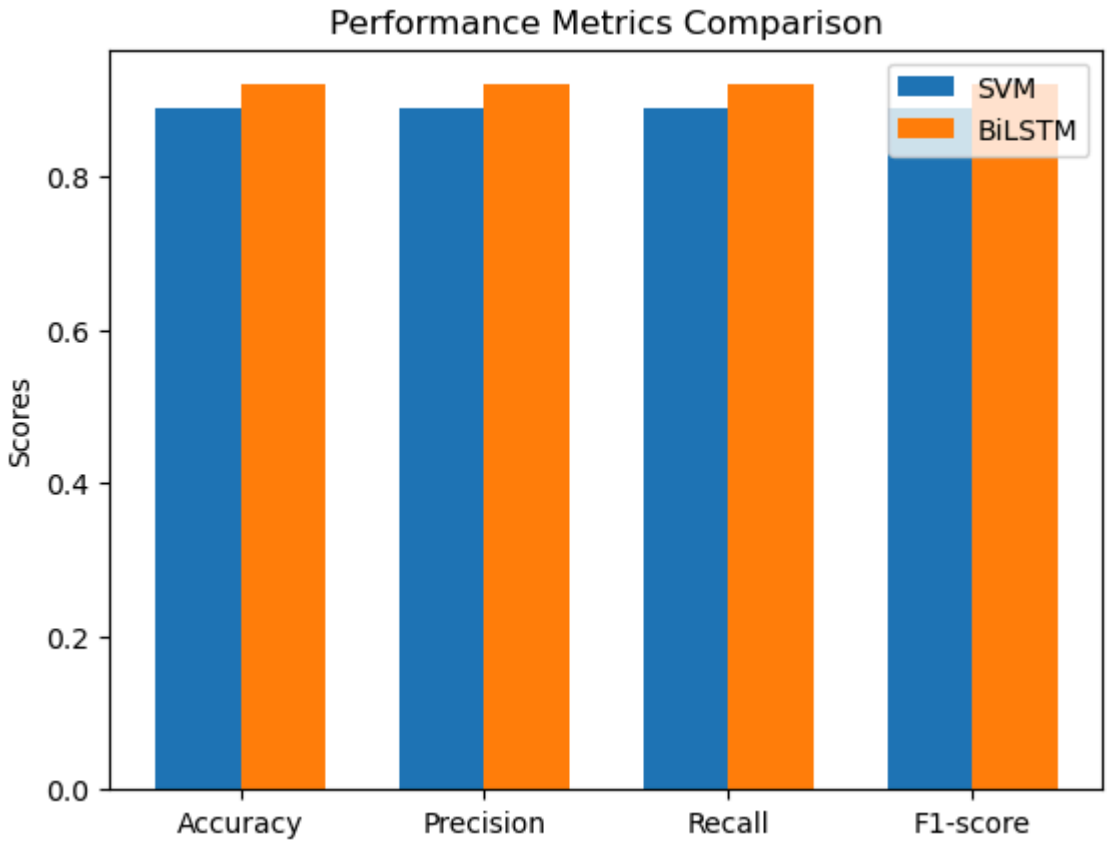
Analysis of Results

The SVM model has demonstrated robust performance in classifying emotions, particularly excelling in precision and recall for certain classes like 'joy' and 'anger'. However, it struggled with more nuanced emotions like 'fear' and 'surprise' and this is likely due to its reliance on the BoW representation, which lacks contextual information. Despite its simplicity and clear interpretability, the SVM model often misclassified 'fear' and 'anger' as 'sadness' when comparing both confusion matrices.

The BiLSTM model, leveraging word embeddings, provided superior overall performance across most metrics, especially in handling subtle emotional nuances. It exhibited higher overall accuracy and balanced performance across all emotion classes, as evident from both the classification report and confusion matrix. The BiLSTM's ability to capture long-term dependencies and bidirectional context contributed to its effectiveness. Although it can capture context and handle noisy data, it still struggled slightly in identifying 'fear' and 'love', thus indicating challenges in distinguishing between subtle emotional expressions in textual data.

In [11]:
```
labels = ['Accuracy', 'Precision', 'Recall', 'F1-score']
svm_metrics = [svm_accuracy, svm_precision, svm_recall, svm_f1]
bilstm_metrics = [bilstm_accuracy, bilstm_precision, bilstm_recall, bilstm_f1]

x = np.arange(len(labels))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, svm_metrics, width, label='SVM')
rects2 = ax.bar(x + width/2, bilstm_metrics, width, label='BiLSTM')
```

```
ax.set_ylabel('Scores')
ax.set_title('Performance Metrics Comparison')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

plt.show()
```



Performance Disparities and Scenarios for Preference

The observed performance disparities highlight the contextual understanding provided by BiLSTM, which is crucial for nuanced tasks like mental health sentiment analysis. In scenarios where computational resources are limited or interpretability is paramount, the SVM model might be preferred. Conversely, for tasks requiring deep contextual understanding and handling of large, complex datasets, the BiLSTM model would be more suitable.

Hypotheses for Performance Disparities

- **Contextual Understanding**: The superior performance of the BiLSTM model can be attributed to its ability to capture context and long-term dependencies in text, which is crucial for accurately classifying emotions.
- **Data representation**: The use of word embeddings in the BiLSTM model allows it to understand semantic similarities and differences between words, providing a richer representation than the BoW approach used in the SVM model.
- **Model Complexity**: The complexity and depth of the BiLSTM model enable it to learn more intricate patterns in the data, leading to better performance, particularly on more challenging and nuanced emotion classes.

Project Summary and Reflections

Learning experience

This project has been an extensive exploration of text classification, specifically within mental health sentiment analysis. It provided an opportunity to deeply analyze both traditional statistical models, such as SVM, and modern deep learning models, like BiLSTM, uncovering their respective strengths and weaknesses. Hands-on experience in data preprocessing, model implementation, and performance evaluation significantly enhanced my understanding of theoretical concepts and practical applications in Natural Language Processing (NLP).

Practicality of Each Model Type and its Potential Applications

The SVM model, known for its simplicity and rapid deployment, proved practical in scenarios prioritizing interpretability over computational complexity. Despite its lower performance compared to BiLSTM, SVM remains valuable where computational resources are constrained (Aggarwal & Zhai, 2012).

Conversely, BiLSTM demonstrated superior capability in capturing intricate emotional nuances within text, emphasizing its practicality in tasks demanding deep contextual comprehension despite higher computational demands (Young et al., 2018).

Both models exhibit diverse real-world applications. SVM excels in quick sentiment analysis for mental health surveys and real-time social media monitoring. BiLSTM, with its advanced learning capabilities, is suited for complex tasks such as detailed emotional analysis in mental health studies, personalized therapy recommendations, and monitoring online forums for emerging mental health trends.

Contributions to Problem Area and Transferabilty to Other Domains

This project contributes to the field of mental health informatics by empirically demonstrating the strengths and limitations of different text classification models. The findings provide valuable insights into how various modelling approaches can be leveraged to improve the accuracy and reliability of sentiment analysis tools (Calvo et al., 2017). These contributions are pivotal for developing better tools to monitor and understand emotional well-being, potentially leading to more informed decisions and timely interventions in mental health care.

Methodologies from this study are adaptable to other domains. The comparative analysis of SVM and BiLSTM can inform tasks like fake news detection, spam filtering, and customer feedback analysis. Techniques in preprocessing, model implementation, and performance evaluation are versatile for different text types and classification challenges (Aggarwal & Zhai, 2012).

Improvements and Future Research Directions

Future enhancements might explore ensemble methods blending statistical and deep learning models' strengths. Advanced preprocessing techniques like context-aware tokenization and dynamic stopword removal could further refine model performance. Expanding dataset diversity and representativeness would enhance generalizability.

Future research could focus on developing hybrid models that integrate the interpretability of statistical models with the contextual understanding of deep learning models. Exploring the use of transfer learning to adapt pre-trained models to specific mental health datasets could further improve performance. Additionally, investigating the ethical implications and potential biases in sentiment analysis models, particularly in sensitive areas like mental health, is crucial for ensuring fair and reliable applications.

References

- Calvo, R. A., Milne, D. N., Hussain, M. S., Christensen, H., & Bidargaddi, N. (2017). Natural language processing in mental health applications using non-clinical texts. Journal of Biomedical Informatics, 69, 1 - 9. doi:10.1016/j.jbi.2017.03.001
- Guntuku, S. C., Yaden, D. B., Kern, M. L., Ungar, L. H., & Eichstaedt, J. C. (2019). Detecting depression and mental illness on social media: An integrative review. Current Opinion in Behavioral Sciences, 31, 82 - 89. doi:10.1016/j.cobeha.2019.07.007
- Aggarwal, C. C., & Zhai, C. (2012). A survey of text classification algorithms. In Mining Text Data (pp. 163-222). Springer.
- Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. IEEE Computational Intelligence Magazine, 13(3), 55-75.