YouTube comments sentiment — LSTM

This notebook loads `YoutubeCommentsDataSet.csv` (columns **Comment**, **Sentiment**), holds out **20%** for a final test set, and splits the remaining **80%** into training and validation. Comments are tokenized and fed through an **Embedding → LSTM** classifier for three-way sentiment (**positive**, **negative**, **neutral**).

Analysis Setup

```
# If imports fail, run once in this environment:
# %pip install pandas scikit-learn tensorflow

import os
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

RANDOM_STATE = 123
np.random.seed(RANDOM_STATE)
tf.random.set_seed(RANDOM_STATE)

print("TensorFlow:", tf.__version__)
```

```
TensorFlow: 2.19.0
```

Preparing Text Data

```
# Load and clean data
DATA_PATH = "/content/YoutubeCommentsDataSet.csv"

df = pd.read_csv(DATA_PATH)
df = df.dropna(subset=["Comment", "Sentiment"]).copy()
df["Comment"] = df["Comment"].astype(str).str.strip()
df = df[df["Comment"].str.len() > 0]
```

```
# Separate two columns into X and Y
X = df["Comment"].values
y = df["Sentiment"].values

# 20% holdout test; remaining 80% split into train + validation
X_trainval, X_test, y_trainval, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=RANDOM_STATE,
    stratify=y,
)

# Of the 80% pool, use 20% as validation → ~64% train / 16% val / 20% test overall
X_train, X_val, y_train, y_val = train_test_split(
    X_trainval,
    y_trainval,
    test_size=0.2,
    random_state=RANDOM_STATE,
    stratify=y_trainval,
)
```

```
# Encode sentiments into numerical values for all datasets
label_encoder = LabelEncoder()
y_train_enc = label_encoder.fit_transform(y_train)
y_val_enc = label_encoder.transform(y_val)
y_test_enc = label_encoder.transform(y_test)

num_classes = len(label_encoder.classes_)
print("Classes:", list(label_encoder.classes_))

MAX_WORDS = 20000
MAX_LEN = 200
```

```
# Add tokenizer to transform sentences to vectors
tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_val_seq = tokenizer.texts_to_sequences(X_val)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Add padding to sentences shorter than the limit
X_train_pad   =   pad_sequences(X_train_seq,   maxlen=MAX_LEN,   padding="post",
truncating="post")
X_val_pad    =    pad_sequences(X_val_seq,    maxlen=MAX_LEN,    padding="post",
truncating="post")
X_test_pad   =   pad_sequences(X_test_seq,   maxlen=MAX_LEN,   padding="post",
truncating="post")

max_token_id = int(max(X_train_pad.max(), X_val_pad.max(), X_test_pad.max()))
vocab_size = min(MAX_WORDS, len(tokenizer.word_index) + 1)
print("Max token id in splits:", max_token_id, "| Embedding input_dim:",
vocab_size)
```

```
Classes: ['negative', 'neutral', 'positive']
Max token id in splits: 19999 | Embedding input_dim: 20000
```

Helper Function

```
# Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, labels, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels,
                yticklabels=labels)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(title)
    plt.show()
```

Model 1: LSTM

```
model = keras.Sequential(name="comment_sentiment_lstm")
model.add(layers.Embedding(input_dim=vocab_size,
```

```
output_dim=128, input_length=MAX_LEN))
model.add(layers.Dropout(0.3))
model.add(layers.LSTM(64))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(num_classes, activation="softmax"))

model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/
core/embedding.py:100: UserWarning: Argument `input_length` is deprecated. Just
remove it.
  warnings.warn(
```

```
Model: "comment_sentiment_lstm"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
dropout (Dropout)	?	0
lstm (LSTM)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0
dense (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
```

```
 Trainable params: 0 (0.00 B)
```

```
 Non-trainable params: 0 (0.00 B)
```

```
# Add early stopping to model
callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=3,
        restore_best_weights=True,
    ),
    ModelCheckpoint(
    filepath='best_model_init.keras',
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1
)
]

history = model.fit(
    X_train_pad,
    y_train_enc,
    validation_data=(X_val_pad, y_val_enc),
    epochs=20,
    batch_size=64,
    callbacks=callbacks,
    verbose=1,
)
```

```
Epoch 1/20
182/184 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6132 - loss: 0.9272
Epoch 1: val_loss improved from None to 0.90625, saving model to
best_model_init.keras

Epoch 1: finished saving model to best_model_init.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.6193 - loss: 0.9150 -
val_accuracy: 0.6220 - val_loss: 0.9063
Epoch 2/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6248 - loss: 0.9067
Epoch 2: val_loss did not improve from 0.90625
184/184 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.6234 - loss: 0.9083 -
val_accuracy: 0.6206 - val_loss: 0.9064
Epoch 3/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6266 - loss: 0.9064
Epoch 3: val_loss improved from 0.90625 to 0.90607, saving model to
```

```
best_model_init.keras

Epoch 3: finished saving model to best_model_init.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.6248 - loss: 0.9077 -
val_accuracy: 0.6220 - val_loss: 0.9061
Epoch 4/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6269 - loss: 0.9035
Epoch 4: val_loss did not improve from 0.90607
184/184 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6257 - loss: 0.9055 -
val_accuracy: 0.6216 - val_loss: 0.9084
Epoch 5/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6276 - loss: 0.9005
Epoch 5: val_loss did not improve from 0.90607
184/184 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.6269 - loss: 0.9018 -
val_accuracy: 0.6220 - val_loss: 0.9146
Epoch 6/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6281 - loss: 0.8985
Epoch 6: val_loss did not improve from 0.90607
184/184 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.6277 - loss: 0.8994 -
val_accuracy: 0.6216 - val_loss: 0.9115
```

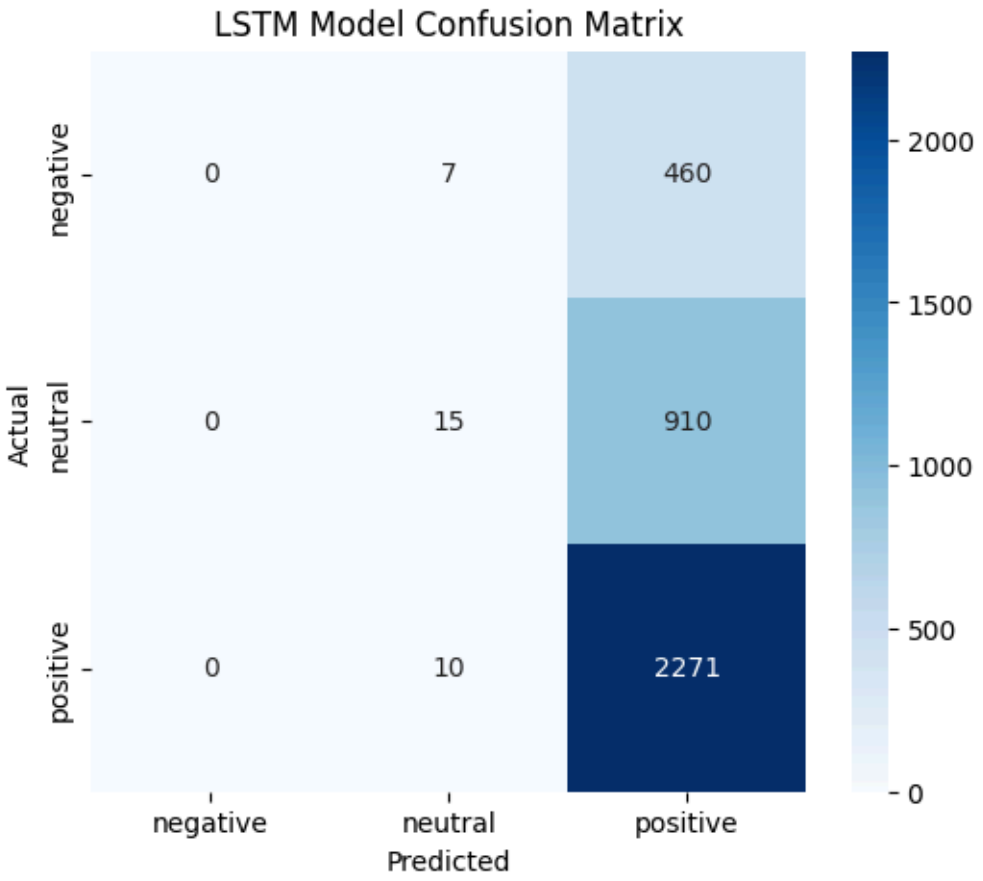
```
model.load_weights('best_model_init.keras')
test_loss, test_acc = model.evaluate(X_test_pad, y_test_enc, verbose=0)
print(f"Test loss: {test_loss:.4f}  |  Test accuracy: {test_acc:.4f}")

y_pred_proba = model.predict(X_test_pad, verbose=0)
y_pred_enc = np.argmax(y_pred_proba, axis=1)

print("\nClassification report (test):")
print(
    classification_report(
        y_test_enc,
        y_pred_enc,
        target_names=label_encoder.classes_,
    )
)
print("Confusion matrix (rows=true, cols=pred):")
print(confusion_matrix(y_test_enc, y_pred_enc))
```

```
Test loss: 0.9039  |  Test accuracy: 0.6224

Classification report (test):
```

```
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       467
     neutral       0.47      0.02      0.03       925
    positive       0.62      1.00      0.77      2281

    accuracy                           0.62      3673
   macro avg       0.36      0.34      0.27      3673
weighted avg       0.51      0.62      0.48      3673

Confusion matrix (rows=true, cols=pred):
[[   0    7  460]
 [   0   15  910]
 [   0   10 2271]]
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
plot_confusion_matrix(
    y_test_enc,
    y_pred_enc,
    label_encoder.classes_,
    title="LSTM Model Confusion Matrix"
)
```




```
# Get class weights for each sentiment
class_weights = compute_class_weight(
    class_weight="balanced",
    classes=np.unique(y_train_enc),
    y=y_train_enc
)

class_weights_dict = dict(enumerate(class_weights))
print("Class Weights:", class_weights_dict)
```

```
Class       Weights:       {0:       np.float64(2.6185383244206775),       1:
np.float64(1.3234234234234235), 2: np.float64(0.5369152046783626)}
```

```
# Add early stopping to model
callbacks = [
    EarlyStopping(
```

```
        monitor="val_loss",
        patience=3,
        restore_best_weights=True,
    ),
    ModelCheckpoint(
    filepath='best_model_weights.keras',
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1
)
]

# Train with class weights
history = model.fit(
    X_train_pad,
    y_train_enc,
    validation_data=(X_val_pad, y_val_enc),
    epochs=20,
    batch_size=64,
    callbacks=callbacks,
    class_weight=class_weights_dict,
    verbose=1,
)
```

```
Epoch 1/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4387 - loss: 1.1178
Epoch 1: val_loss improved from None to 1.09230, saving model to
best_model_weights.keras

Epoch 1: finished saving model to best_model_weights.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.3810 - loss: 1.1047 -
val_accuracy: 0.1313 - val_loss: 1.0923
Epoch 2/20
181/184 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.4178 - loss: 1.0904
Epoch 2: val_loss did not improve from 1.09230
184/184 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.3683 - loss: 1.0961 -
val_accuracy: 0.1313 - val_loss: 1.0967
Epoch 3/20
182/184 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4429 - loss: 1.0881
Epoch 3: val_loss did not improve from 1.09230
184/184 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.3923 - loss: 1.0929 -
```

```
val_accuracy: 0.1313 - val_loss: 1.0976
Epoch 4/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.4162 - loss: 1.0864
Epoch 4: val_loss did not improve from 1.09230
184/184 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.3592 - loss: 1.0904 -
val_accuracy: 0.1310 - val_loss: 1.0993
```

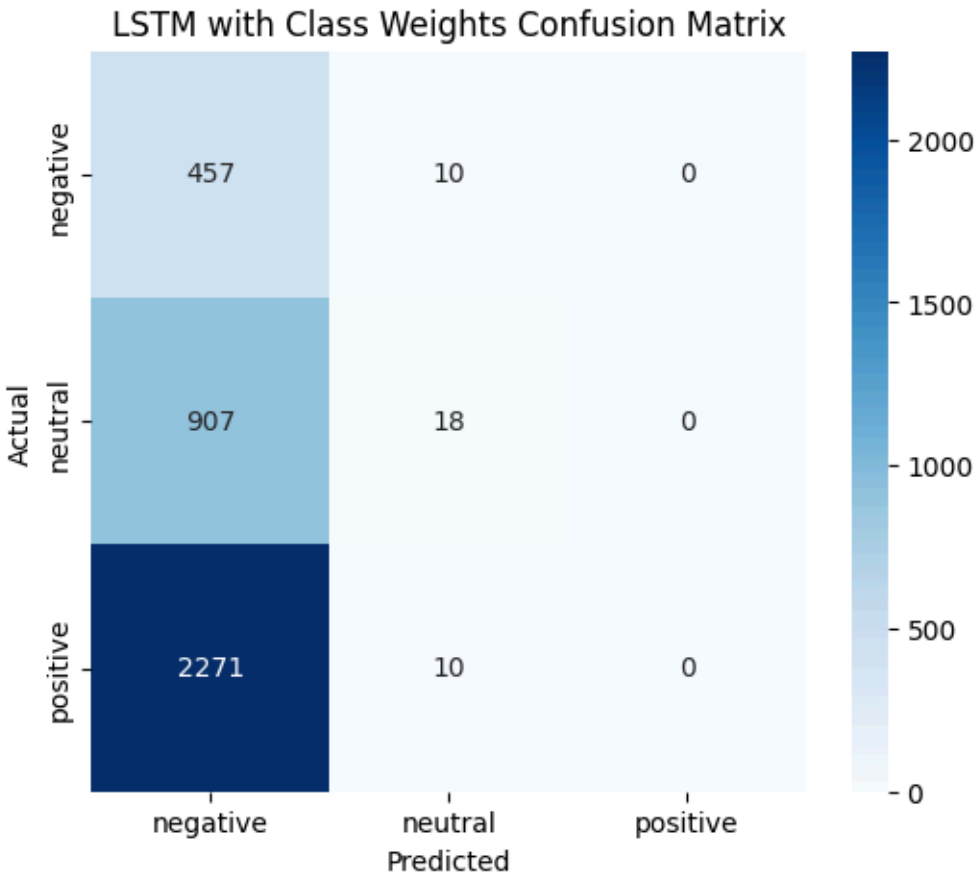
```
model.load_weights('best_model_weights.keras')

# Evaluate model trained on previous weights
test_loss, test_acc = model.evaluate(X_test_pad, y_test_enc, verbose=0)
print(f"Test loss: {test_loss:.4f}  |  Test accuracy: {test_acc:.4f}")

y_pred_proba = model.predict(X_test_pad, verbose=0)
y_pred_enc = np.argmax(y_pred_proba, axis=1)

print("\nClassification report (test):")
print(
    classification_report(
        y_test_enc,
        y_pred_enc,
        target_names=label_encoder.classes_,
    )
)
print("Confusion matrix (rows=true, cols=pred):")
print(confusion_matrix(y_test_enc, y_pred_enc))
```

```
Test loss: 1.0916  |  Test accuracy: 0.1293

Classification report (test):
              precision    recall  f1-score   support

    negative       0.13      0.98      0.22       467
     neutral       0.47      0.02      0.04       925
    positive       0.00      0.00      0.00      2281

    accuracy                           0.13      3673
   macro avg       0.20      0.33      0.09      3673
weighted avg       0.14      0.13      0.04      3673

Confusion matrix (rows=true, cols=pred):
[[ 457   10    0]
```

```
 [ 907   18    0]
 [2271   10    0]]
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
plot_confusion_matrix(
    y_test_enc,
    y_pred_enc,
    label_encoder.classes_,
    title="LSTM with Class Weights Confusion Matrix"
)
```



Bidirectional LSTM Model

```
# Adjust model so it can read sentence forward and backward
model = keras.Sequential(name="comment_sentiment_bidi_lstm")
model.add(layers.Embedding(input_dim=vocab_size,
output_dim=128, input_length=MAX_LEN))
model.add(layers.Dropout(0.3))
model.add(layers.Bidirectional(layers.LSTM(64)))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(num_classes, activation="softmax"))

model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/
core/embedding.py:100: UserWarning: Argument `input_length` is deprecated. Just
remove it.
  warnings.warn(
```

```
Model: "comment_sentiment_bidi_lstm"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ embedding_2 (Embedding)              │ ?                           │   0 (unbuilt) │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_4 (Dropout)                  │ ?                           │             0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ bidirectional (Bidirectional)        │ ?                           │   0 (unbuilt) │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_5 (Dropout)                  │ ?                           │             0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ ?                           │   0 (unbuilt) │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
 Total params: 0 (0.00 B)
```

```
 Trainable params: 0 (0.00 B)
```

```
 Non-trainable params: 0 (0.00 B)
```

```
# Add early stopping to model
callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=3,
        restore_best_weights=True,
    ),
    ModelCheckpoint(
    filepath='best_model_bidi.keras',
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1
```

```
)
]

history = model.fit(
    X_train_pad,
    y_train_enc,
    validation_data=(X_val_pad, y_val_enc),
    epochs=20,
    batch_size=64,
    callbacks=callbacks,
    verbose=1,
)
```

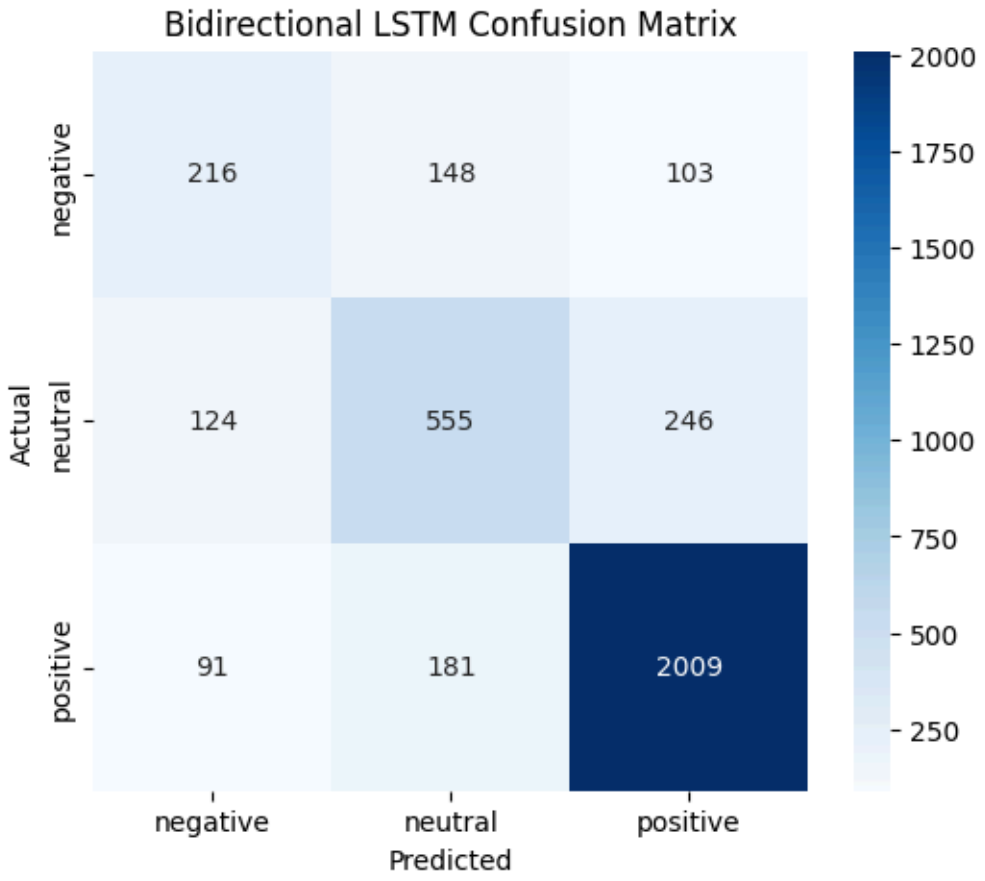
```
Epoch 1/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6381 - loss: 0.8470
Epoch 1: val_loss improved from None to 0.63929, saving model to
best_model_bidi.keras

Epoch 1: finished saving model to best_model_bidi.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 7s 24ms/step - accuracy: 0.6772 - loss: 0.7528 -
val_accuracy: 0.7166 - val_loss: 0.6393
Epoch 2/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7702 - loss: 0.5367
Epoch 2: val_loss improved from 0.63929 to 0.58529, saving model to
best_model_bidi.keras

Epoch 2: finished saving model to best_model_bidi.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 7s 35ms/step - accuracy: 0.7954 - loss: 0.4891 -
val_accuracy: 0.7438 - val_loss: 0.5853
Epoch 3/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8676 - loss: 0.3486
Epoch 3: val_loss did not improve from 0.58529
184/184 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.8806 - loss: 0.3211 -
val_accuracy: 0.7411 - val_loss: 0.6869
Epoch 4/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9074 - loss: 0.2499
Epoch 4: val_loss did not improve from 0.58529
184/184 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9155 - loss: 0.2361 -
val_accuracy: 0.7387 - val_loss: 0.8689
Epoch 5/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9446 - loss: 0.1574
Epoch 5: val_loss did not improve from 0.58529
```

```
184/184 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - accuracy: 0.9495 - loss: 0.1480 -
val_accuracy: 0.7271 - val_loss: 0.9687
```

```
model.load_weights('best_model_bidi.keras')

test_loss, test_acc = model.evaluate(X_test_pad, y_test_enc, verbose=0)
print(f"Test loss: {test_loss:.4f}  |  Test accuracy: {test_acc:.4f}")

y_pred_proba = model.predict(X_test_pad, verbose=0)
y_pred_enc = np.argmax(y_pred_proba, axis=1)

print("\nClassification report (test):")
print(
    classification_report(
        y_test_enc,
        y_pred_enc,
        target_names=label_encoder.classes_,
    )
)
print("Confusion matrix (rows=true, cols=pred):")
print(confusion_matrix(y_test_enc, y_pred_enc))
```

```
Test loss: 0.5735  |  Test accuracy: 0.7569

Classification report (test):
              precision    recall  f1-score   support

    negative       0.50      0.46      0.48       467
     neutral       0.63      0.60      0.61       925
    positive       0.85      0.88      0.87      2281

    accuracy                           0.76      3673
   macro avg       0.66      0.65      0.65      3673
weighted avg       0.75      0.76      0.75      3673

Confusion matrix (rows=true, cols=pred):
[[ 216  148  103]
 [ 124  555  246]
 [  91  181 2009]]
```

```
plot_confusion_matrix(
    y_test_enc,
    y_pred_enc,
    label_encoder.classes_,
    title="Bidirectional LSTM Confusion Matrix"
)
```



Using GloVe Embeddings Map

```
# Download the GloVe embeddings
!wget https://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2026-04-21 16:23:21--  https://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443...
connected.
```

```
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2026-04-21 16:23:22--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.
zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|
171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.08MB/s    in 2m 40s

2026-04-21 16:26:03 (5.12 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
EMBED_DIM = 100
embeddings_index = {}

# Load embedding vectors into dict
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

print(f"Found {len(embeddings_index)} word vectors.")
```

```
Found 400000 word vectors.
```

```
# Initialize matrix with zeros
embedding_matrix = np.zeros((vocab_size, EMBED_DIM))

# Adds GloVe embeddings to embedding matrix
for word, i in tokenizer.word_index.items():
```

```
    if i < vocab_size:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

```
model = keras.Sequential(name="GloVe_sentiment_lstm")

# The Embedding layer now uses our GloVe matrix
model.add(layers.Embedding(
    input_dim=vocab_size,
    output_dim=EMBED_DIM,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False
))

model.add(layers.SpatialDropout1D(0.3))
model.add(layers.Bidirectional(layers.LSTM(64, return_sequences=True)))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(num_classes, activation="softmax"))

model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "GloVe_sentiment_lstm"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_3 (Embedding)         │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ spatial_dropout1d               │ ?                      │             0 │
│ (SpatialDropout1D)              │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional_1 (Bidirectional) │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_max_pooling1d            │ ?                      │             0 │
│ (GlobalMaxPooling1D)            │                        │               │
```

```
├─────────────────────────────────┼────────────────────────┼────────────────┤
│ dense_3 (Dense)                 │ ?                      │   0 (unbuilt) │
├─────────────────────────────────┼────────────────────────┼────────────────┤
│ dropout_6 (Dropout)             │ ?                      │              0 │
├─────────────────────────────────┼────────────────────────┼────────────────┤
│ dense_4 (Dense)                 │ ?                      │   0 (unbuilt) │
└─────────────────────────────────┴────────────────────────┴────────────────┘
```

```
 Total params: 0 (0.00 B)
```

```
 Trainable params: 0 (0.00 B)
```

```
 Non-trainable params: 0 (0.00 B)
```

```
# Add early stopping, checkpoint to model
callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=3,
        restore_best_weights=True,
    ),
    ModelCheckpoint(
    filepath='best_model_glove.keras',
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1
)
]

# Lessen weight of negative classification
smooth_weights_dict = class_weights_dict.copy()
smooth_weights_dict[0] = 0.5 * smooth_weights_dict[0]

history = model.fit(
    X_train_pad,
    y_train_enc,
    validation_data=(X_val_pad, y_val_enc),
    epochs=20,
    batch_size=64,
    callbacks=callbacks,
```

```
    class_weight=smooth_weights_dict,
    verbose=1,
)
```

```
Epoch 1/20
182/184 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5935 - loss: 0.8175
Epoch 1: val_loss improved from None to 0.65337, saving model to
best_model_glove.keras

Epoch 1: finished saving model to best_model_glove.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 11s 40ms/step - accuracy: 0.6407 - loss: 0.7498 -
val_accuracy: 0.7234 - val_loss: 0.6534
Epoch 2/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.6898 - loss: 0.6432
Epoch 2: val_loss improved from 0.65337 to 0.58468, saving model to
best_model_glove.keras

Epoch 2: finished saving model to best_model_glove.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.7054 - loss: 0.6187 -
val_accuracy: 0.7564 - val_loss: 0.5847
Epoch 3/20
183/184 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7094 - loss: 0.5927
Epoch 3: val_loss improved from 0.58468 to 0.56224, saving model to
best_model_glove.keras

Epoch 3: finished saving model to best_model_glove.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 7s 41ms/step - accuracy: 0.7220 - loss: 0.5739 -
val_accuracy: 0.7605 - val_loss: 0.5622
Epoch 4/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7251 - loss: 0.5669
Epoch 4: val_loss improved from 0.56224 to 0.55240, saving model to
best_model_glove.keras

Epoch 4: finished saving model to best_model_glove.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.7332 - loss: 0.5535 -
val_accuracy: 0.7666 - val_loss: 0.5524
Epoch 5/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.7375 - loss: 0.5423
Epoch 5: val_loss improved from 0.55240 to 0.54420, saving model to
best_model_glove.keras

Epoch 5: finished saving model to best_model_glove.keras
```

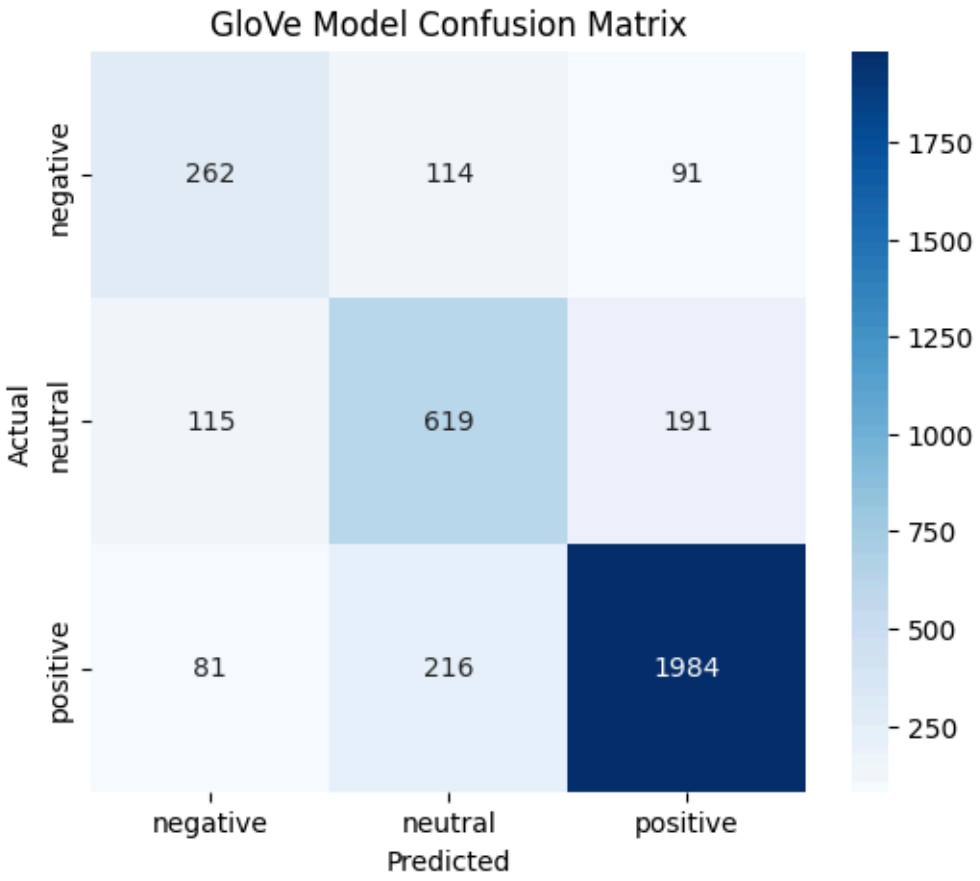
```
184/184 ━━━━━━━━━━━━━━━━━━━━ 16s 86ms/step - accuracy: 0.7477 - loss: 0.5320 -
val_accuracy: 0.7710 - val_loss: 0.5442
Epoch 6/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7478 - loss: 0.5285
Epoch 6: val_loss improved from 0.54420 to 0.54275, saving model to
best_model_glove.keras

Epoch 6: finished saving model to best_model_glove.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 12s 42ms/step - accuracy: 0.7555 - loss: 0.5168 -
val_accuracy: 0.7795 - val_loss: 0.5427
Epoch 7/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7506 - loss: 0.5171
Epoch 7: val_loss improved from 0.54275 to 0.53406, saving model to
best_model_glove.keras

Epoch 7: finished saving model to best_model_glove.keras
184/184 ━━━━━━━━━━━━━━━━━━━━ 8s 46ms/step - accuracy: 0.7582 - loss: 0.5034 -
val_accuracy: 0.7761 - val_loss: 0.5341
Epoch 8/20
182/184 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7518 - loss: 0.5122
Epoch 8: val_loss did not improve from 0.53406
184/184 ━━━━━━━━━━━━━━━━━━━━ 6s 21ms/step - accuracy: 0.7606 - loss: 0.4978 -
val_accuracy: 0.7782 - val_loss: 0.5355
Epoch 9/20
184/184 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7634 - loss: 0.4904
Epoch 9: val_loss did not improve from 0.53406
184/184 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.7725 - loss: 0.4792 -
val_accuracy: 0.7771 - val_loss: 0.5382
Epoch 10/20
182/184 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7686 - loss: 0.4853
Epoch 10: val_loss did not improve from 0.53406
184/184 ━━━━━━━━━━━━━━━━━━━━ 5s 28ms/step - accuracy: 0.7768 - loss: 0.4750 -
val_accuracy: 0.7748 - val_loss: 0.5424
```

```
model.load_weights('best_model_glove.keras')

test_loss, test_acc = model.evaluate(X_test_pad, y_test_enc, verbose=0)
print(f"Test loss: {test_loss:.4f}  |  Test accuracy: {test_acc:.4f}")

y_pred_proba = model.predict(X_test_pad, verbose=0)
y_pred_enc = np.argmax(y_pred_proba, axis=1)
```

```
print("\nClassification report (test):")
print(
    classification_report(
        y_test_enc,
        y_pred_enc,
        target_names=label_encoder.classes_,
    )
)
print("Confusion matrix (rows=true, cols=pred):")
print(confusion_matrix(y_test_enc, y_pred_enc))
```

```
Test loss: 0.5160  |  Test accuracy: 0.7800

Classification report (test):
              precision    recall  f1-score   support

    negative       0.57      0.56      0.57       467
     neutral       0.65      0.67      0.66       925
    positive       0.88      0.87      0.87      2281

    accuracy                           0.78      3673
   macro avg       0.70      0.70      0.70      3673
weighted avg       0.78      0.78      0.78      3673

Confusion matrix (rows=true, cols=pred):
[[ 262  114   91]
 [ 115  619  191]
 [  81  216 1984]]
```

```
plot_confusion_matrix(
    y_test_enc,
    y_pred_enc,
    label_encoder.classes_,
    title="GloVe Model Confusion Matrix"
)
```




```
# # 1. Unfreeze the Embedding layer (it's the first layer, index 0)
# model.layers[0].trainable = True

# # 2. Re-compile the model to apply the change
# # Use a much smaller learning rate (1e-5 instead of 1e-3)
# model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-5),
#               loss='sparse_categorical_crossentropy',
#               metrics=['accuracy'])

# print("Starting fine-tuning...")
# # 3. Continue training for a few more epochs
# history_fine_tune = model.fit(
#     X_train_pad, y_train_enc,
#     epochs=5,
#     validation_data=(X_val_pad, y_val_enc),
#     class_weight=smooth_weights_dict,
#     callbacks=callbacks,
#     verbose=1,
```

```
#     batch_size=32
# )
```

```
Starting fine-tuning...
Epoch 1/5
368/368 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7714 - loss: 0.4967
Epoch 1: val_loss did not improve from 0.53406
368/368 ━━━━━━━━━━━━━━━━━━━━ 12s 26ms/step - accuracy: 0.7779 - loss: 0.4812 -
val_accuracy: 0.7768 - val_loss: 0.5407
Epoch 2/5
368/368 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7688 - loss: 0.4976
Epoch 2: val_loss did not improve from 0.53406
368/368 ━━━━━━━━━━━━━━━━━━━━ 8s 22ms/step - accuracy: 0.7758 - loss: 0.4834 -
val_accuracy: 0.7768 - val_loss: 0.5404
Epoch 3/5
366/368 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7648 - loss: 0.4925
Epoch 3: val_loss did not improve from 0.53406
368/368 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.7743 - loss: 0.4758 -
val_accuracy: 0.7754 - val_loss: 0.5417
```

```
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Notes: -We had four models: 1 - Standard LSTM 2 - LSTM using class weights 3 - Bi-directional LSTM 4 - LSTM using GloVe embedding vectors

Model 1 skewed towards classifying almost everything as positive, since it is the most frequent sentiment. Model 2 overweighted negative results and classified almost everything as negative. Model 3 did a much better job at gaining context with the bi-directional layer, and Model 4 performed the best. Model 4 uses embeddings from GloVe instead of learning them from the model, and it also adds a MaxPooling layer and softens the weight of negative results. While accuracy only improves by a few percentage points in Model 4, it achieves a greater balance in recall and precision between all three sentiments.