

Model Development Phase Template

Date	24 November 2024
Team ID	SWTID1727420425
Project Title	Analysis amazon cell phone review with nlp technique
Maximum Marks	10 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code for analyzing Amazon cell phone reviews using NLP techniques will be showcased in the future through a screenshot. The model validation and evaluation report will provide a summary of the performance, including training and validation metrics for multiple models. These performance metrics will be presented through respective screenshots, highlighting key indicators such as accuracy, precision, recall, and F1 score to assess the effectiveness of the NLP models in classifying and analyzing customer sentiments from the reviews.

Initial Model Training Code (5 marks):

```

• Model Building

! pip install keras-tuner --upgrade
!pip install scikeras tensorflow scikit-learn

Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (3.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (24.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (2.32.3)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (3.12.1)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (0.13.1)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras->keras-tuner) (0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2024.8.30)
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.10/dist-packages (from optree->keras->keras-tuner) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras->keras-tuner) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras->keras-tuner) (2.18.0)
Requirement already satisfied: mdurl<=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras->keras-tuner) (0.1.2)
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
   129.1/129.1 kB 2.7 MB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)

```

```
[ ] import keras_tuner as kt
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Conv1D, LSTM, Bidirectional, Embedding, MaxPooling1D, Dropout, Flatten
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.optimizers import Adam
```

• Data Pre processing

```
[ ] # Selecting features and target
features = ['rating_x', 'verified', 'title_x', 'body', 'brand', 'price', 'originalPrice']
target = 'helpfulVotes'
```

```
▶ # Clean text function
def clean_text(text):
    text = text.lower()
    text = re.sub(r'\W', ' ', text) # Remove special characters
    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text) # Remove single characters
    text = re.sub(r'\s+', ' ', text) # Remove multiple spaces
    return text
```

[+ Code](#)[+ Text](#)

```
[ ] # Apply cleaning to text features
merged_data['body'] = merged_data['body'].fillna('').apply(clean_text)
merged_data['title_x'] = merged_data['title_x'].fillna('').apply(clean_text)
```

```
[ ] # Filter data and drop missing rows
filtered_data = merged_data[features + [target]].dropna()
```

```
[ ] max_sequence_length = 100
    X_train_pad = pad_sequences(X_train_seq, maxlen=max_sequence_length)
    X_test_pad = pad_sequences(X_test_seq, maxlen=max_sequence_length)

[ ] # Binarizing the target
    y_train = y_train.values
    y_test = y_test.values

▶ # Define class weights to handle imbalance
  class_weights = class_weight.compute_class_weight(
      class_weight='balanced',
      classes=np.unique(y_train),
      y=y_train
  )
  class_weights_dict = {i: class_weights[i] for i in range(len(class_weights))}
```

Model Validation and Evaluation Report (5 marks):

Model	Model code	Model Classification

[illegible]

```
[1] # Evaluate the ANN model
test_loss, test_accuracy = ann_best_model.evaluate(X_test_pad, y_test)
print(f"ANN Model Test Accuracy: {test_accuracy:.4f}")

170/170 — 0s 2ms/step - accuracy: 1.0000 - loss: 3.7855e-24
ANN Model Test Accuracy: 1.0000

# Generate predictions and classification report
y_pred_probs_ann = ann_best_model.predict(X_test_pad)
y_pred_ann = (y_pred_probs_ann > 0.5).astype(int)
print("\nANN Classification Report:")
print(classification_report(y_test, y_pred_ann))

170/170 — 1s 3ms/step

ANN Classification Report:
              precision    recall  f1-score   support

     1         1.00         1.00         1.00         5416

 accuracy          1.00          1.00          1.00         5416
 macro avg          1.00          1.00          1.00         5416
weighted avg          1.00          1.00          1.00         5416
```

```

1 def build_resnet_model():
2     """Resnet18"""
3     model = Sequential()
4     model.add(layers.InputLayer(input_dim=28*28, output_shape=(1, 1, 28, 28), input_length=(sequence_length, sequence_width)))
5     model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), border_mode='same', activation='relu'))
6     model.add(layers.MaxPooling2D(pool_size=(2, 2)))
7     model.add(layers.Flatten())
8     model.add(layers.Dense(units=120, kernel_initializer='he_normal', activation='relu'))
9     model.add(layers.Dense(units=80, kernel_initializer='he_normal', activation='relu'))
10    model.add(layers.Dense(units=10, kernel_initializer='he_normal', activation='softmax'))
11    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
12                  jit_compile=True,
13                  tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
14    return model
15
16 # Training and testing on model
17 time_start = time.time()
18 model = build_resnet_model()
19 compile_model(model, num_epochs, max_trials,
20               patience_per_trial,
21               directory=os.path.join(
22                   os.getcwd(), 'logs'
23               ),
24               callbacks=[callbacks])
25
26 # Print the results
27 print('Results: %s' % results)
28
29 # Print 5 complete (min dev std) val accuracy: 1.0
30 best_val_accuracy = 0.0
31 total_elapsed_time = 0.0
32 for i in range(1, num_epochs + 1):
33     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
34                 jit_compile=True,
35                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
36     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
37             shuffle=True, verbose=0, callbacks=[callbacks],
38             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
39     best_model_path = os.path.join(directory, 'best_model.h5')
40     if best_val_accuracy < model.get_loss(test_data, test_labels):
41         model.save(best_model_path)
42         best_val_accuracy = model.get_loss(test_data, test_labels)
43     total_elapsed_time += time.time() - time_start
44     time_start = time.time()
45
46 # Print 5 complete (min dev std) val accuracy: 1.0
47 best_val_accuracy = 0.0
48 total_elapsed_time = 0.0
49 for i in range(1, num_epochs + 1):
50     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
51                 jit_compile=True,
52                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
53     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
54             shuffle=True, verbose=0, callbacks=[callbacks],
55             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
56     best_model_path = os.path.join(directory, 'best_model.h5')
57     if best_val_accuracy < model.get_loss(test_data, test_labels):
58         model.save(best_model_path)
59         best_val_accuracy = model.get_loss(test_data, test_labels)
60     total_elapsed_time += time.time() - time_start
61     time_start = time.time()
62
63 # Print 5 complete (min dev std) val accuracy: 1.0
64 best_val_accuracy = 0.0
65 total_elapsed_time = 0.0
66 for i in range(1, num_epochs + 1):
67     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
68                 jit_compile=True,
69                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
70     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
71             shuffle=True, verbose=0, callbacks=[callbacks],
72             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
73     best_model_path = os.path.join(directory, 'best_model.h5')
74     if best_val_accuracy < model.get_loss(test_data, test_labels):
75         model.save(best_model_path)
76         best_val_accuracy = model.get_loss(test_data, test_labels)
77     total_elapsed_time += time.time() - time_start
78     time_start = time.time()
79
80 # Print 5 complete (min dev std) val accuracy: 1.0
81 best_val_accuracy = 0.0
82 total_elapsed_time = 0.0
83 for i in range(1, num_epochs + 1):
84     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
85                 jit_compile=True,
86                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
87     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
88             shuffle=True, verbose=0, callbacks=[callbacks],
89             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
90     best_model_path = os.path.join(directory, 'best_model.h5')
91     if best_val_accuracy < model.get_loss(test_data, test_labels):
92         model.save(best_model_path)
93         best_val_accuracy = model.get_loss(test_data, test_labels)
94     total_elapsed_time += time.time() - time_start
95     time_start = time.time()
96
97 # Print 5 complete (min dev std) val accuracy: 1.0
98 best_val_accuracy = 0.0
99 total_elapsed_time = 0.0
100 for i in range(1, num_epochs + 1):
101     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
102                 jit_compile=True,
103                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
104     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
105             shuffle=True, verbose=0, callbacks=[callbacks],
106             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
107     best_model_path = os.path.join(directory, 'best_model.h5')
108     if best_val_accuracy < model.get_loss(test_data, test_labels):
109         model.save(best_model_path)
110         best_val_accuracy = model.get_loss(test_data, test_labels)
111     total_elapsed_time += time.time() - time_start
112     time_start = time.time()
113
114 # Print 5 complete (min dev std) val accuracy: 1.0
115 best_val_accuracy = 0.0
116 total_elapsed_time = 0.0
117 for i in range(1, num_epochs + 1):
118     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
119                 jit_compile=True,
120                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
121     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
122             shuffle=True, verbose=0, callbacks=[callbacks],
123             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
124     best_model_path = os.path.join(directory, 'best_model.h5')
125     if best_val_accuracy < model.get_loss(test_data, test_labels):
126         model.save(best_model_path)
127         best_val_accuracy = model.get_loss(test_data, test_labels)
128     total_elapsed_time += time.time() - time_start
129     time_start = time.time()
130
131 # Print 5 complete (min dev std) val accuracy: 1.0
132 best_val_accuracy = 0.0
133 total_elapsed_time = 0.0
134 for i in range(1, num_epochs + 1):
135     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
136                 jit_compile=True,
137                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
138     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
139             shuffle=True, verbose=0, callbacks=[callbacks],
140             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
141     best_model_path = os.path.join(directory, 'best_model.h5')
142     if best_val_accuracy < model.get_loss(test_data, test_labels):
143         model.save(best_model_path)
144         best_val_accuracy = model.get_loss(test_data, test_labels)
145     total_elapsed_time += time.time() - time_start
146     time_start = time.time()
147
148 # Print 5 complete (min dev std) val accuracy: 1.0
149 best_val_accuracy = 0.0
150 total_elapsed_time = 0.0
151 for i in range(1, num_epochs + 1):
152     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
153                 jit_compile=True,
154                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
155     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
156             shuffle=True, verbose=0, callbacks=[callbacks],
157             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
158     best_model_path = os.path.join(directory, 'best_model.h5')
159     if best_val_accuracy < model.get_loss(test_data, test_labels):
160         model.save(best_model_path)
161         best_val_accuracy = model.get_loss(test_data, test_labels)
162     total_elapsed_time += time.time() - time_start
163     time_start = time.time()
164
165 # Print 5 complete (min dev std) val accuracy: 1.0
166 best_val_accuracy = 0.0
167 total_elapsed_time = 0.0
168 for i in range(1, num_epochs + 1):
169     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
170                 jit_compile=True,
171                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
172     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
173             shuffle=True, verbose=0, callbacks=[callbacks],
174             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
175     best_model_path = os.path.join(directory, 'best_model.h5')
176     if best_val_accuracy < model.get_loss(test_data, test_labels):
177         model.save(best_model_path)
178         best_val_accuracy = model.get_loss(test_data, test_labels)
179     total_elapsed_time += time.time() - time_start
180     time_start = time.time()
181
182 # Print 5 complete (min dev std) val accuracy: 1.0
183 best_val_accuracy = 0.0
184 total_elapsed_time = 0.0
185 for i in range(1, num_epochs + 1):
186     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
187                 jit_compile=True,
188                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
189     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
190             shuffle=True, verbose=0, callbacks=[callbacks],
191             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
192     best_model_path = os.path.join(directory, 'best_model.h5')
193     if best_val_accuracy < model.get_loss(test_data, test_labels):
194         model.save(best_model_path)
195         best_val_accuracy = model.get_loss(test_data, test_labels)
196     total_elapsed_time += time.time() - time_start
197     time_start = time.time()
198
199 # Print 5 complete (min dev std) val accuracy: 1.0
200 best_val_accuracy = 0.0
201 total_elapsed_time = 0.0
202 for i in range(1, num_epochs + 1):
203     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
204                 jit_compile=True,
205                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
206     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
207             shuffle=True, verbose=0, callbacks=[callbacks],
208             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
209     best_model_path = os.path.join(directory, 'best_model.h5')
210     if best_val_accuracy < model.get_loss(test_data, test_labels):
211         model.save(best_model_path)
212         best_val_accuracy = model.get_loss(test_data, test_labels)
213     total_elapsed_time += time.time() - time_start
214     time_start = time.time()
215
216 # Print 5 complete (min dev std) val accuracy: 1.0
217 best_val_accuracy = 0.0
218 total_elapsed_time = 0.0
219 for i in range(1, num_epochs + 1):
220     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
221                 jit_compile=True,
222                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
223     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
224             shuffle=True, verbose=0, callbacks=[callbacks],
225             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
226     best_model_path = os.path.join(directory, 'best_model.h5')
227     if best_val_accuracy < model.get_loss(test_data, test_labels):
228         model.save(best_model_path)
229         best_val_accuracy = model.get_loss(test_data, test_labels)
230     total_elapsed_time += time.time() - time_start
231     time_start = time.time()
232
233 # Print 5 complete (min dev std) val accuracy: 1.0
234 best_val_accuracy = 0.0
235 total_elapsed_time = 0.0
236 for i in range(1, num_epochs + 1):
237     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
238                 jit_compile=True,
239                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
240     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
241             shuffle=True, verbose=0, callbacks=[callbacks],
242             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
243     best_model_path = os.path.join(directory, 'best_model.h5')
244     if best_val_accuracy < model.get_loss(test_data, test_labels):
245         model.save(best_model_path)
246         best_val_accuracy = model.get_loss(test_data, test_labels)
247     total_elapsed_time += time.time() - time_start
248     time_start = time.time()
249
250 # Print 5 complete (min dev std) val accuracy: 1.0
251 best_val_accuracy = 0.0
252 total_elapsed_time = 0.0
253 for i in range(1, num_epochs + 1):
254     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'],
255                 jit_compile=True,
256                 tensorboard_callback=TensorBoard(log_dir='./logs', write_graph=True, write_images=True))
257     model.fit(train_data_generator, validation_data=(test_data, test_labels), class_weight=dict(weights_dict),
258             shuffle=True, verbose=0, callbacks=[callbacks],
259             validation_data=(test_data, test_labels), class_weight=dict(weights_dict))
260     best_model
```

```
[ ] test_loss, test_accuracy = cnn_best_model.evaluate(X_test_pad, y_test)
print(f"CNN Model Test Accuracy: {test_accuracy:.4f}")

170/170 ————— 3s 15ms/step - accuracy: 1.0000 - loss: 2.8217e-11
CNN Model Test Accuracy: 1.0000

# Generate predictions and classification report
y_pred_probs_cnn = cnn_best_model.predict(X_test_pad)
y_pred_cnn = (y_pred_probs_cnn > 0.5).astype(int)
print("CNN Classification Report:")
print(classification_report(y_test, y_pred_cnn))

170/170 ————— 4s 22ms/step

CNN Classification Report:
              precision    recall  f1-score   support

     1           1.00        1.00        1.00         5416
 accuracy           1.00        1.00        1.00         5416
 macro avg           1.00        1.00        1.00         5416
 weighted avg          1.00        1.00        1.00         5416
```

```

3 train

4 def build_lstm_model():
5     model = Sequential()
6     model.add(LSTM(128, input_shape=(input_length, input_channels)))
7     model.add(Dense(128))
8     model.add(Dense(1))
9     model.compile(loss='mse', optimizer='adam')
10    return model

11 # Hyperparameter tuning with Keras Tuner
12 tuner = kt.RandomSearch(
13     build_lstm_model,
14     objective='val_accuracy',
15     max_trials=100,
16     execution_timeout_minutes=10,
17     directory='./lstm_tuning',
18     project_name='lstm_model')

19 # Fit the model
20 tuner.search_space_builder.set_searchable_params('lstm_model', {'input_length': 10, 'input_channels': 1})
21 tuner.search()
22 tuner.get_best_models(num_models=1)

```

```
[ ] Evaluate the model
test_loss, test_accuracy = lstm_best_model.evaluate(X_test_pad, y_test)
print("Test Accuracy: {}".format(test_accuracy))

170/170 ——— 11s 66ms/step - accuracy: 1.0000 - loss: 3.1954e-09
Test Accuracy: 1.0000

c Generate predictions and classification report
y_pred_probs = lstm_best_model.predict(X_test_pad)
y_pred = (y_pred_probs > 0.5).astype(int)
print("Classification Report:")
print(classification_report(y_test, y_pred))

170/170 ——— 11s 66ms/step

Classification Report:
          precision    recall  f1-score   support

     1         1.00      1.00      1.00         5416
 accuracy          1.00      1.00      1.00         5416
 macro avg          1.00      1.00      1.00         5416
 weighted avg          1.00      1.00      1.00         5416
```

BiLSTM

```
[ ] # Split training data into train and validation sets
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train_pad, y_train, test_size=0.2, random_state=42)

[ ] # Run the tuner
tuner.search(X_train_split, y_train_split, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Trial 5 Complete (on 200 steps)
val_accuracy: 1.0
Best val accuracy so far: 1.0
Total elapsed time: 00h 43m 26s

[ ] # Get the best model
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
lstm_best_model = tuner.get_best_model(num_trials=1)[0]

/usr/local/lib/python3.10/dist-packages/keras/src/rnn/layer.py:733: UserWarning: Skipping variable loading for non-savable load_out_variables(weights_store.get(inner_path))

[ ] # Train the best model
lstm_best_model.fit(X_train_pad, y_train, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Epoch 1/5
4/7/2027 130s 200ms/step - accuracy: 1.0000 - loss: 1.0000e-06 - val_accuracy: 1.0000 - val_loss: 1.0000e-06
Epoch 2/5
4/7/2027 141s 200ms/step - accuracy: 1.0000 - loss: 1.0000e-06 - val_accuracy: 1.0000 - val_loss: 1.0000e-06
Epoch 3/5
----- completed at 11:03 AM -----

[ ] # Get the best model
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
lstm_best_model = tuner.get_best_model(num_trials=1)[0]

/usr/local/lib/python3.10/dist-packages/keras/src/rnn/layer.py:733: UserWarning: Skipping variable loading for optimizer 'adam', because non-savable load_out_variables(weights_store.get(inner_path))

[ ] # Train the best model
lstm_best_model.fit(X_train_pad, y_train, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Epoch 1/5
4/7/2027 130s 200ms/step - accuracy: 1.0000 - loss: 1.0000e-06 - val_accuracy: 1.0000 - val_loss: 1.0000e-06
Epoch 2/5
4/7/2027 141s 200ms/step - accuracy: 1.0000 - loss: 1.0000e-06 - val_accuracy: 1.0000 - val_loss: 1.0000e-06
Epoch 3/5
4/7/2027 141s 200ms/step - accuracy: 1.0000 - loss: 9.3750e-06 - val_accuracy: 1.0000 - val_loss: 4.6791e-06
Epoch 4/5
4/7/2027 144s 200ms/step - accuracy: 1.0000 - loss: 7.8812e-06 - val_accuracy: 1.0000 - val_loss: 3.7050e-06
Epoch 5/5
4/7/2027 146s 200ms/step - accuracy: 1.0000 - loss: 5.7160e-06 - val_accuracy: 1.0000 - val_loss: 3.1061e-06
callbacks.CallbackList.history: history of the lstm_best_model

[ ] lstm_best_model.summary()

Model: "sequential"
Layer (type) Output Shape Param #
-----
embedding (Embedding) (None, 1, 1) 1
```

```
[ ] # Build BiLSTM Model
def build_bilstm_model(hps):
    model = Sequential()
    model.add(Embedding(input_dim=vocab.get_vocab('embedding').get_vocab_size('tokens'), output_dim=hps.get('embedding_dim', 64, 256, stepsize), input_length=max_sequence_length))
    model.add(Bidirectional(LSTM(units=hps.get('lstm_units', 32, 256, stepsize), return_sequences=True)))
    model.add(Dense(units=hps.get('dense_units', 128, 256, stepsize), activation='relu'))
    model.add(Dense(units=hps.get('dense_units', 128, 256, stepsize), activation='relu'))
    model.compile(optimizer=hps.get('optimizer', 'adam', 1e-4, 1e-2, sampling='100'), loss='categorical_crossentropy')
    return model

[ ] # Train and evaluate BiLSTM model
tuner_bilstm = KerasTuner(
    build_bilstm_model,
    objective='val_accuracy',
    num_trials=10,
    executions_per_trial=1,
    directory='bilstm_training',
    project_name='bilstm_model'
)

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument 'input_length' is deprecated. Just remove it
warnings.warn()
```

```
[ ] # Train BiLSTM model
lstm_best_model = tuner_bilstm.search(X_train_split, y_train_split, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)
bilstm_best_model = tuner_bilstm.get_best_model(num_trials=1)[0]
bilstm_best_model.fit(X_train_pad, y_train, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Trial 5 Complete (on 200 steps)
val_accuracy: 1.0
Best val accuracy so far: 1.0
Total elapsed time: 00h 40m 40s

Epoch 1/5
4/7/2027 130s 200ms/step - accuracy: 1.0000 - loss: 4.2500e-07 - val_accuracy: 1.0000 - val_loss: 5.5000e-06
Epoch 2/5
4/7/2027 137s 200ms/step - accuracy: 1.0000 - loss: 1.0000e-06 - val_accuracy: 1.0000 - val_loss: 2.2000e-06
Epoch 3/5
4/7/2027 139s 200ms/step - accuracy: 1.0000 - loss: 1.0000e-06 - val_accuracy: 1.0000 - val_loss: 1.0000e-06
Epoch 4/5
4/7/2027 140s 200ms/step - accuracy: 1.0000 - loss: 3.7500e-06 - val_accuracy: 1.0000 - val_loss: 1.0000e-06
Epoch 5/5
4/7/2027 140s 200ms/step - accuracy: 1.0000 - loss: 2.5500e-06 - val_accuracy: 1.0000 - val_loss: 6.0000e-06
callbacks.CallbackList.history: history of the bilstm_best_model

[ ] bilstm_best_model.summary()

Model: "sequential"
Layer (type) Output Shape Param #
-----
embedding (Embedding) (None, 1, 1) 1
bidirectional (Bidirectional) (None, 1) 1
dense (Dense) (None, 1) 1
dense_1 (Dense) (None, 1) 1
Total params: 10 (20.87 MB)
Trainable params: 10 (20.87 MB)
Non-Trainable params: 0 (0.00 MB)
Optimizer params: 10 (20.87 MB)

[ ] # Evaluate BiLSTM Model
test_loss, test_accuracy = bilstm_best_model.evaluate(X_test_pad, y_test)
print(f"BiLSTM Model Test Accuracy: {test_accuracy:.4f}")

170/170 3s 20ms/step - accuracy: 1.0000 - loss: 7.9719e-10
BiLSTM Model Test Accuracy: 1.0000

[ ] # Generate predictions and classification report
y_pred_probs_bilstm = bilstm_best_model.predict(X_test_pad)
y_pred_bilstm = (y_pred_probs_bilstm > 0.5).astype(int)
print("BiLSTM Classification Report:")
print(classification_report(y_test, y_pred_bilstm))

170/170 4s 25ms/step

BiLSTM Classification Report:
precision recall f1-score support
1 1.00 1.00 1.00 5416
accuracy 1.00 1.00 1.00 5416
macro avg 1.00 1.00 1.00 5416
weighted avg 1.00 1.00 1.00 5416
```

```
[ ] # Evaluate BiLSTM Model
test_loss, test_accuracy = bilstm_best_model.evaluate(X_test_pad, y_test)
print(f"BiLSTM Model Test Accuracy: {test_accuracy:.4f}")

170/170 3s 20ms/step - accuracy: 1.0000 - loss: 7.9719e-10
BiLSTM Model Test Accuracy: 1.0000

[ ] # Generate predictions and classification report
y_pred_probs_bilstm = bilstm_best_model.predict(X_test_pad)
y_pred_bilstm = (y_pred_probs_bilstm > 0.5).astype(int)
print("BiLSTM Classification Report:")
print(classification_report(y_test, y_pred_bilstm))

170/170 4s 25ms/step

BiLSTM Classification Report:
precision recall f1-score support
1 1.00 1.00 1.00 5416
accuracy 1.00 1.00 1.00 5416
macro avg 1.00 1.00 1.00 5416
weighted avg 1.00 1.00 1.00 5416
```