Including nesseccery libraries:

```
import pandas as pd
import numpy as np
import re
import seaborn as sns
import matplotlib.pyplot as plt
import time
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt_tab')
from matplotlib import style
style.use('ggplot')
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay,log_loss
```

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

Data loading:

```
covid_hate = pd.read_csv('COVID-hate.csv')
```

```
covid_hate.head()
```

	Tweet ID	Text	label
0	1242553623260868608	Are we still allowed to quote ancient Chinese ...	0
1	1246508137638580225	@mamacat2u @VBeltiz More power to you! This C...	0
2	1233468243534372865	CNBC: WHO, Tedros reiterated that the virus co...	0
3	1243626072387747841	"The heightened racism experienced by Asian co...	1
4	1225611530978217989	Coronavirus and Nepali in China: KP Oli has di...	0

```
covid_hate.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Tweet ID  2290 non-null   int64 
 1   Text      2290 non-null   object
 2   label     2290 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 53.8+ KB
```

Data preprocessing:

```
print(covid_hate['Text'].iloc[0],"\n")
print(covid_hate['Text'].iloc[1],"\n")
print(covid_hate['Text'].iloc[2],"\n")
print(covid_hate['Text'].iloc[3],"\n")
print(covid_hate['Text'].iloc[4],"\n")
```

```
Are we still allowed to quote ancient Chinese proverbs, or is that racist? #RacismIsAVirus

@mamacat2u @VBeltiz More power to you!  This Chinese virus thing have really shown us who are the crazies and low IQ people are.  I

CNBC: WHO, Tedros reiterated that the virus could still turn into a pandemic. He urged against fear and panic, adding, "our greatest

"The heightened racism experienced by Asian communities is surprising to many people because beliefs in racial progress are widespre

Coronavirus and Nepali in China: KP Oli has directed officials to bring back Nepali in Wuhan, China and Keep them s… https://t.co/u
```

```python
def data_preprocessing(data):
    # Initialize the lemmatizer and stop words
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    # Convert to lowercase
    data = data.lower()

    # Remove non-alphabetic characters
    data = re.sub(r'[^a-zA-Z\s]', '', data)

    # Remove extra spaces
    data = re.sub(r'\s+', ' ', data).strip()

    # Tokenize the data
    data_tokens = word_tokenize(data)

    # Lemmatize and remove stop words
    filtered_data = [lemmatizer.lemmatize(w) for w in data_tokens if w not in stop_words]

    # Return the processed data as a single string
    return " ".join(filtered_data)
```

```python
covid_hate.Text = covid_hate['Text'].apply(data_preprocessing)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-163a4c4aaf60> in <cell line: 0>()
----> 1 covid_hate.Text = covid_hate['Text'].apply(data_preprocessing)

NameError: name 'covid_hate' is not defined
```

```python
covid_hate = covid_hate.drop_duplicates('Text')
```

```python
print(covid_hate['Text'].iloc[0],"\n")
print(covid_hate['Text'].iloc[1],"\n")
print(covid_hate['Text'].iloc[2],"\n")
print(covid_hate['Text'].iloc[3],"\n")
print(covid_hate['Text'].iloc[4],"\n")
```

```
still allowed quote ancient chinese proverb racist racismisavirus

mamacatu vbeltiz power chinese virus thing really shown u crazy low iq people went sam club costco morning store line wrapped arounc

cnbc tedros reiterated virus could still turn pandemic urged fear panic adding greatest enemy right virus fear rumor stigma httpstc

heightened racism experienced asian community surprising many people belief racial progress widespread american society yalesoms mwl

coronavirus nepali china kp oli directed official bring back nepali wuhan china keep httpstcouhiewvyz
```

Data Visualizing:

```python
covid_hate.info()
```

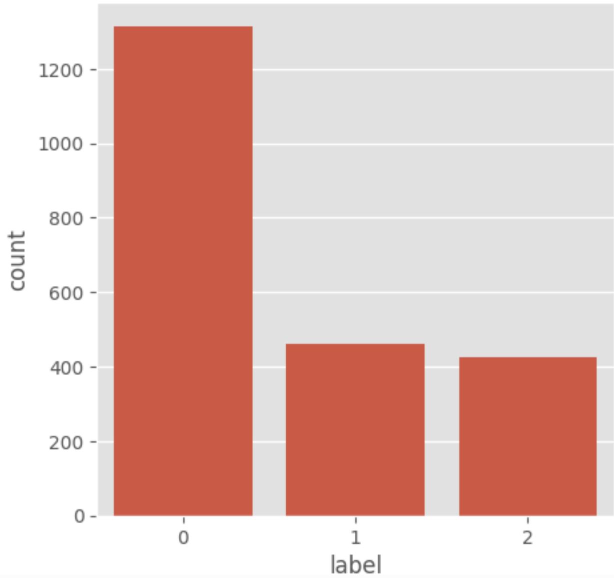
```
<class 'pandas.core.frame.DataFrame'>
Index: 2199 entries, 0 to 2289
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Tweet ID  2199 non-null   int64 
 1   Text      2199 non-null   object
 2   label     2199 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 68.7+ KB
```

```
covid_hate['label'].value_counts()
```

label	count
0	1314
1	461
2	424

```
fig = plt.figure(figsize = (5,5))
sns.countplot(x = 'label',data = covid_hate)
```

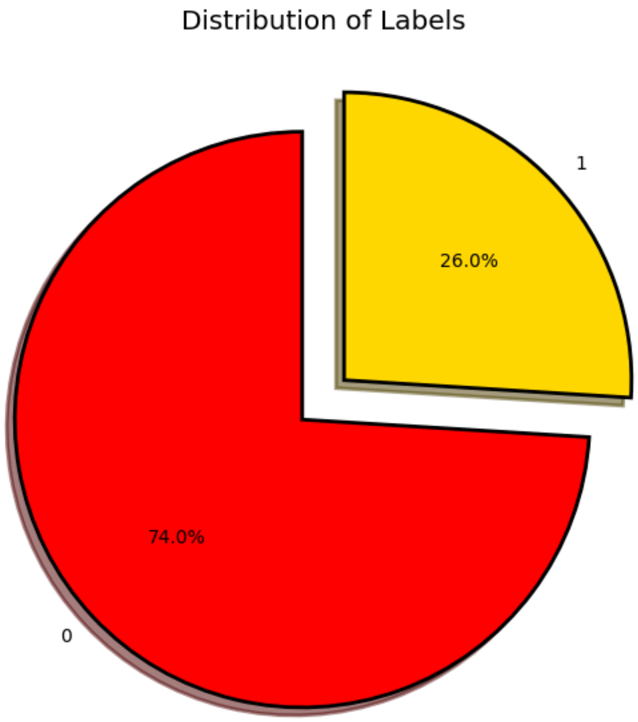
<Axes: xlabel='label', ylabel='count'>




```
covid_hate = covid_hate[covid_hate['label'] != 2]
```

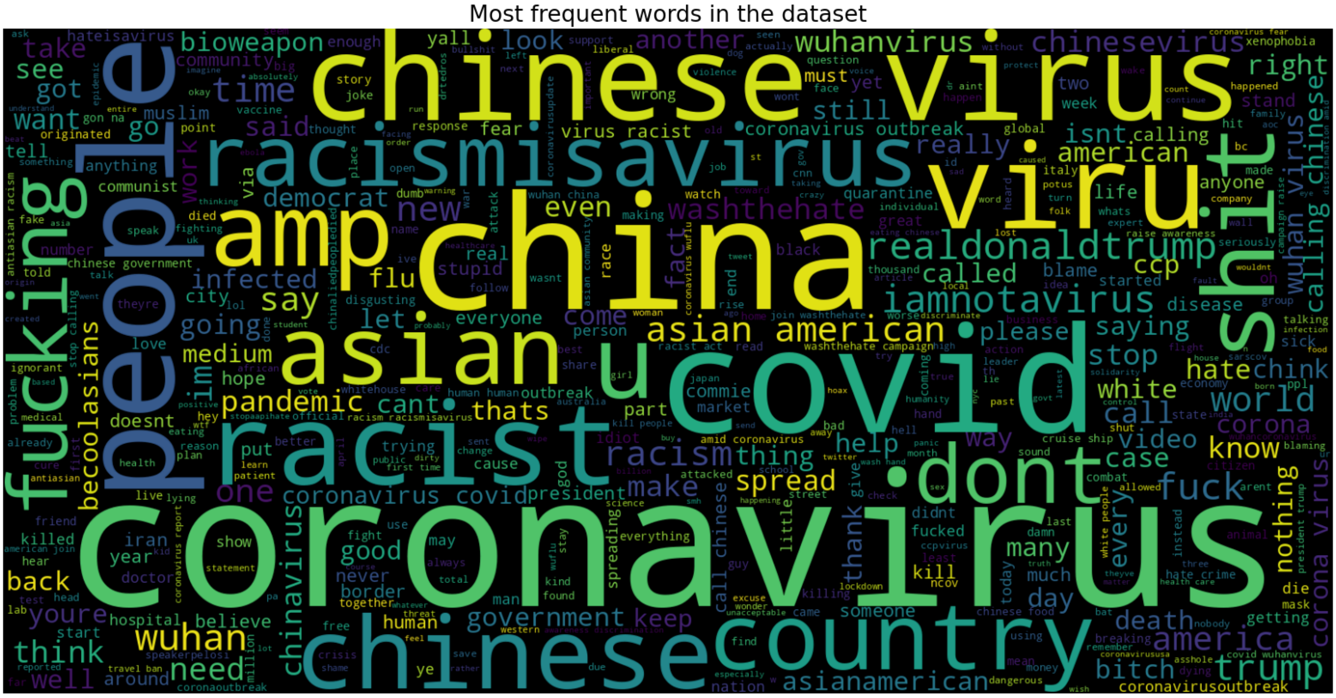
```
fig = plt.figure(figsize = (7,7))
colors = ("red","gold")
wp = {'linewidth' : 2,'edgecolor':"black"}
tags = covid_hate['label'].value_counts()
explode = (0.1,0.1)
tags.plot(kind = 'pie',autopct = '%1.1f%%',shadow = True,colors = colors,startangle = 90,
         wedgeprops = wp,explode = explode,label = '')
plt.title('Distribution of Labels')
```

```
Text(0.5, 1.0, 'Distribution of Labels')
```




```
text = ' '.join([word for word in covid_hate['Text']])
plt.figure(figsize = (20,15),facecolor = 'None')
wordcloud = WordCloud(max_words = 500,width = 1600,height= 800).generate(text)
plt.imshow(wordcloud,interpolation = 'bilinear')
plt.axis('off')
plt.title('Most frequent words in the dataset',fontsize = 19)
plt.show()
```

Most frequent words in the dataset



Vectorization:

```
vect = TfidfVectorizer(ngram_range = (1,2)).fit(covid_hate['Text'])
```

```
feature_names = vect.get_feature_names_out()
```

```
vect = TfidfVectorizer(ngram_range = (1,3)).fit(covid_hate['Text'])
```

```
feature_names = vect.get_feature_names_out()
print("Number of features: {}\n".format(len(feature_names)))
print("First 20 fetures: \n{}".format(feature_names[:20]))
```

```
Number of features: 52794

First 20 fetures:
['aaaaaaaaahhhhhhh' 'aaaaaaaaahhhhhhh going' 'aaaaaaaaahhhhhhh going die'
 'aac' 'aac cancel' 'aac cancel tournament' 'aaccstatement'
 'aaccstatement asianamericans' 'aaccstatement asianamericans solidarity'
 'aaccstatement iamnotavirus' 'aaccstatement iamnotavirus racismisavirus'
 'aajtak' 'aajtak india' 'aajtak india today' 'aap' 'aap bhi'
 'aap bhi band' 'aapi' 'aapi community' 'aapi community covid']
```

Train-Test split

```
x = covid_hate['Text']
y = covid_hate['label']
X = vect.transform(x);
```

```
from sklearn.model_selection import train_test_split
```

```
# Split the dataset into training (60%) and combined validation+test set (40%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
```

```
# Further split the combined validation+test set into validation (50%) and test (50%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Verify the sizes of the splits
print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Training set size: 1065
Validation set size: 355
Test set size: 355
```

logistic Regression Model

```
model = LogisticRegression()
model.fit(X_train,y_train)
model_train = model.predict(X_train)
model_pred = model.predict(X_val)
model_train_acc = accuracy_score(model_train,y_train)
model_acc = accuracy_score(model_pred,y_val)
print("Train accuracy: {:.2f}%".format(model_train_acc*100))
print("Validation accuracy: {:.2f}%".format(model_acc*100))
```

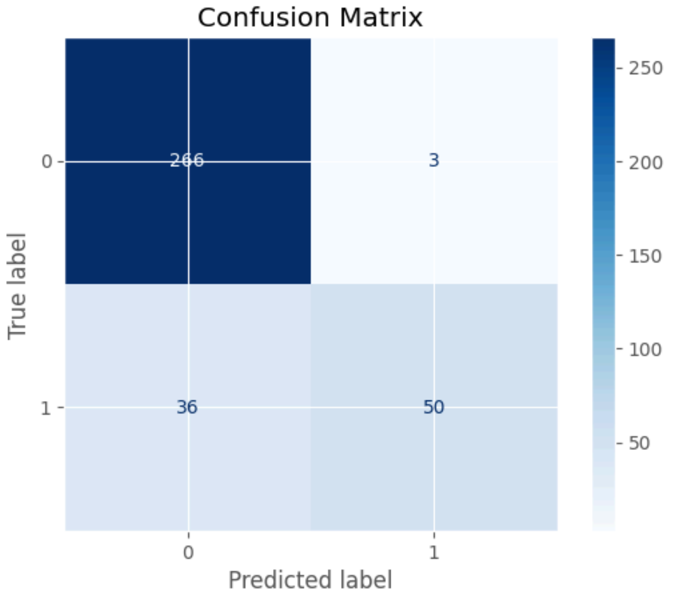
```
Train accuracy: 82.44%
Validation accuracy: 78.03%
```

```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')


#Define the parameter grid for GridSearchCV
param_grid = {'C': [500, 10, 1.0, 0.1, 0.01], 'solver': ['newton-cg', 'lbfgs', 'liblinear']}

#Create the GridSearchCV object
grid = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)

#Measure training time
start_time = time.time()
grid.fit(X_train, y_train)
end_time = time.time()
training_time1 = end_time - start_time

#Best model after grid search
best_model = grid.best_estimator_

#Measure inference time on a subset of the test data
sample_size = 355  # You can adjust this size based on your requirements
sample_x_test = X_test[:sample_size]

start_inference_time = time.time()
_ = best_model.predict(sample_x_test)
end_inference_time = time.time()
inference_time1 = end_inference_time - start_inference_time

# Training and validation accuracy
train_predictions = best_model.predict(X_train)
val_predictions = best_model.predict(X_val)

training_accuracy1 = accuracy_score(y_train, train_predictions)
validation_accuracy1 = accuracy_score(y_val, val_predictions)

# Training loss
train_proba = best_model.predict_proba(X_train)
training_loss1 = log_loss(y_train, train_proba)

# Model complexity (number of parameters)
num_parameters1 = best_model.coef_.size + best_model.intercept_.size

# Print results
print(f"Best Cross Validation score: {grid.best_score_:.2f}")
print(f"Best Parameters: {grid.best_params_}")
print(f"Training time: {training_time1:.2f} seconds")
print(f"Inference time on {sample_size} samples: {inference_time1:.2f} seconds")
print(f"Number of trainable parameters: {num_parameters1}")
print(f"Training loss: {training_loss1:.4f}")
print(f"Training accuracy: {training_accuracy1 * 100:.2f}%")
print(f"Validation accuracy: {validation_accuracy1 * 100:.2f}%")
```

```
Best Cross Validation score: 0.85
Best Parameters: {'C': 500, 'solver': 'lbfgs'}
Training time: 29.26 seconds
Inference time on 355 samples: 0.00 seconds
Number of trainable parameters: 52795
Training loss: 0.0046
Training accuracy: 100.00%
Validation accuracy: 89.01%
```

```
# Get predictions for the validation set using the best model
y_pred = best_model.predict(X_val)

# Compute the confusion matrix
cm = confusion_matrix(y_val, y_pred, labels=best_model.classes_)

# Create a ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_model.classes_)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title('Confusion Matrix')
plt.show()
```

```
<Figure size 1000x700 with 0 Axes>
```



Random Forest Model:

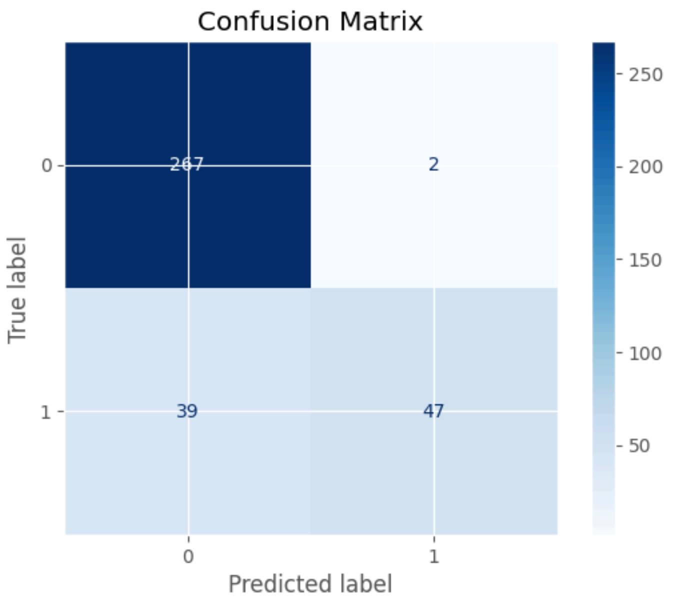
```
# Initialize the RandomForestClassifier with the specified parameters
rf_model = RandomForestClassifier(max_depth=150, n_estimators=19, random_state=10)

# Measure training time
start_time = time.time()
rf_model.fit(X_train, y_train)
end_time = time.time()
training_time2 = end_time - start_time

# Measure inference time on a subset of the test data
sample_size = 355
sample_x_val = X_val[:sample_size]

start_inference_time = time.time()
_ = rf_model.predict(sample_x_test)
end_inference_time = time.time()
inference_time2 = end_inference_time - start_inference_time

# Training accuracy
rf_model_train = rf_model.predict(X_train)
training_accuracy2 = accuracy_score(y_train, rf_model_train)

# Validation accuracy
rf_model_val = rf_model.predict(X_val)
validation_accuracy2 = accuracy_score(y_val, rf_model_val)

# Training loss
```

```
train_proba = rf_model.predict_proba(X_train)
training_loss2 = log_loss(y_train, train_proba)

# Model complexity (number of parameters)
# For RandomForest, model complexity can be estimated by the number of trees and their depth.
num_trees = len(rf_model.estimators_)
average_depth = np.mean([tree.tree_.max_depth for tree in rf_model.estimators_])
model_complexity = f"Number of trees: {num_trees}, Average tree depth: {average_depth:.2f}"

# Print results
print(f"Training time: {training_time2:.2f} seconds")
print(f"Inference time on {sample_size} samples: {inference_time2:.2f} seconds")
print(f"Model complexity: {model_complexity}")
print(f"Training loss: {training_loss2:.4f}")
print(f"Training accuracy: {training_accuracy2 * 100:.2f}%")
print(f"Validation accuracy: {validation_accuracy2 * 100:.2f}%")
```

```
Training time: 1.06 seconds
Inference time on 355 samples: 0.01 seconds
Model complexity: Number of trees: 19, Average tree depth: 121.74
Training loss: 0.0750
Training accuracy: 99.62%
Validation accuracy: 88.45%
```

```
# Compute confusion matrix
y_pred = rf_model.predict(X_val)
cm = confusion_matrix(y_val, y_pred)

# Create and display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
plt.figure(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title('Confusion Matrix')
plt.show()
```

```
<Figure size 1000x700 with 0 Axes>
```



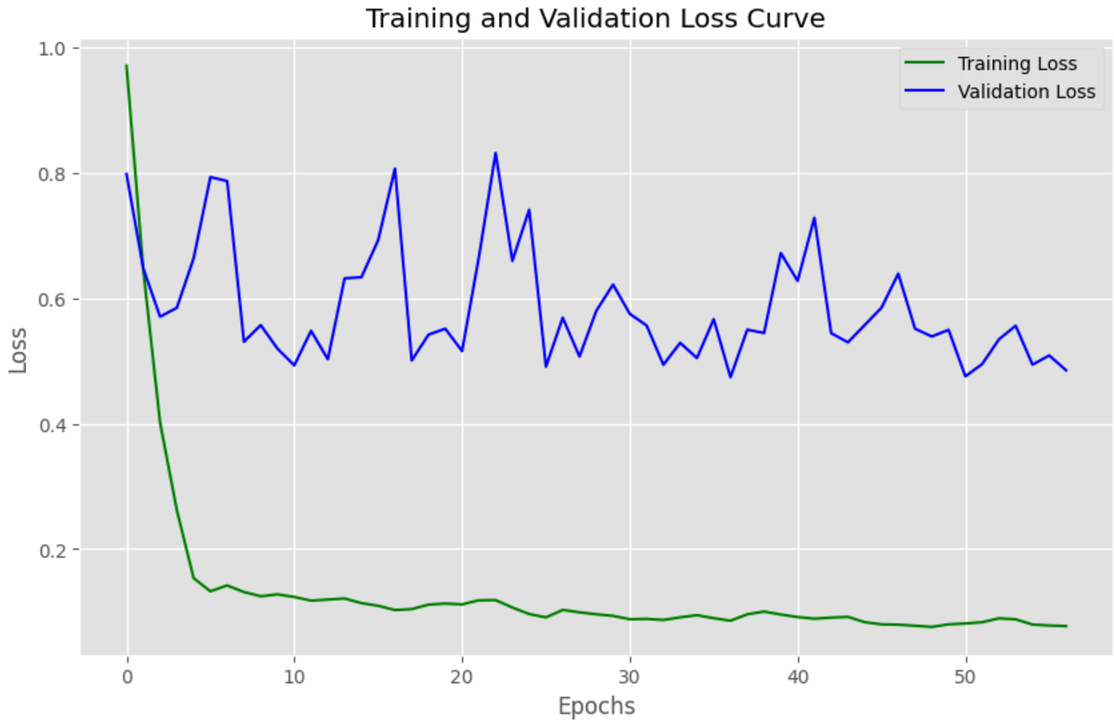
Neural Network

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score

# Define the model
model2 = Sequential([
    Dense(256, activation='relu', name='L1', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(128, activation='relu', name='L2', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(64, activation='relu', name='L3', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(32, activation='relu', name='L4', kernel_regularizer=l2(0.001)),
```

```
    Dropout(0.3),
    Dense(2, activation='softmax', name='Output')
])

# Compile the model
model2.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    optimizer=tf.keras.optimizers.Adam(0.001),
    metrics=['accuracy']
)

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)

# Measure training time
start_time = time.time()
history = model2.fit(X_train, y_train, epochs=200, validation_split=0.2, callbacks=[early_stopping])
end_time = time.time()
training_time3 = end_time - start_time

# Measure inference time on a subset of the test data
sample_size = 355  # You can adjust this size based on your requirements
sample_x_val = X_val[:sample_size]

start_inference_time = time.time()
_ = model2.predict(sample_x_val)
end_inference_time = time.time()
inference_time3 = end_inference_time - start_inference_time

# Get model complexity (number of trainable parameters)
num_parameters = model2.count_params()

# Evaluate the model on training and validation data
train_loss3, train_accuracy3 = model2.evaluate(X_train, y_train, verbose=0)
val_loss3, val_accuracy3 = model2.evaluate(X_val, y_val, verbose=0)

# Print the results
print(f"Training time: {training_time3:.2f} seconds")
print(f"Inference time on {sample_size} samples: {inference_time3:.2f} seconds")
print(f"Number of trainable parameters: {num_parameters}")
print(f"Training loss: {train_loss3:.4f}")
print(f"Training accuracy: {train_accuracy3 * 100:.2f}%")
print(f"Validation loss: {val_loss3:.4f}")
print(f"Validation accuracy: {val_accuracy3 * 100:.2f}%")
```

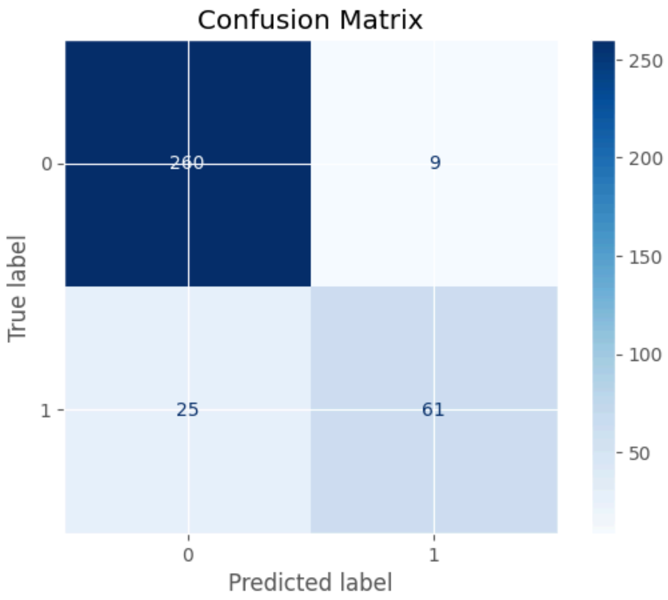
```
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 450ms/step - accuracy: 1.0000 - loss: 0.0876 - val_accuracy: 0.8920 - val_loss: 0.5344
Epoch 54/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 452ms/step - accuracy: 1.0000 - loss: 0.0884 - val_accuracy: 0.8920 - val_loss: 0.5561
Epoch 55/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 447ms/step - accuracy: 1.0000 - loss: 0.0805 - val_accuracy: 0.9108 - val_loss: 0.4941
Epoch 56/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 447ms/step - accuracy: 0.9997 - loss: 0.0771 - val_accuracy: 0.9108 - val_loss: 0.5087
Epoch 57/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 446ms/step - accuracy: 0.9991 - loss: 0.0766 - val_accuracy: 0.8967 - val_loss: 0.4851
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step
Training time: 1032.62 seconds
Inference time on 355 samples: 0.68 seconds
Number of trainable parameters: 13558818
Training loss: 0.1588
Training accuracy: 98.31%
Validation loss: 0.4358
Validation accuracy: 90.42%
```

```python
import matplotlib.pyplot as plt

# Extract loss values from the history object
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot both training and validation loss curves
plt.figure(figsize=(10, 6))
plt.plot(train_loss, label='Training Loss', color='g')
plt.plot(val_loss, label='Validation Loss', color='b')
plt.title('Training and Validation Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```




```python
# Make predictions on the validation data
y_pred = np.argmax(model2.predict(X_val), axis=1)

# Compute the confusion matrix
cm = confusion_matrix(y_val, y_pred)

# Create a ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
<Figure size 1000x700 with 0 Axes>
```



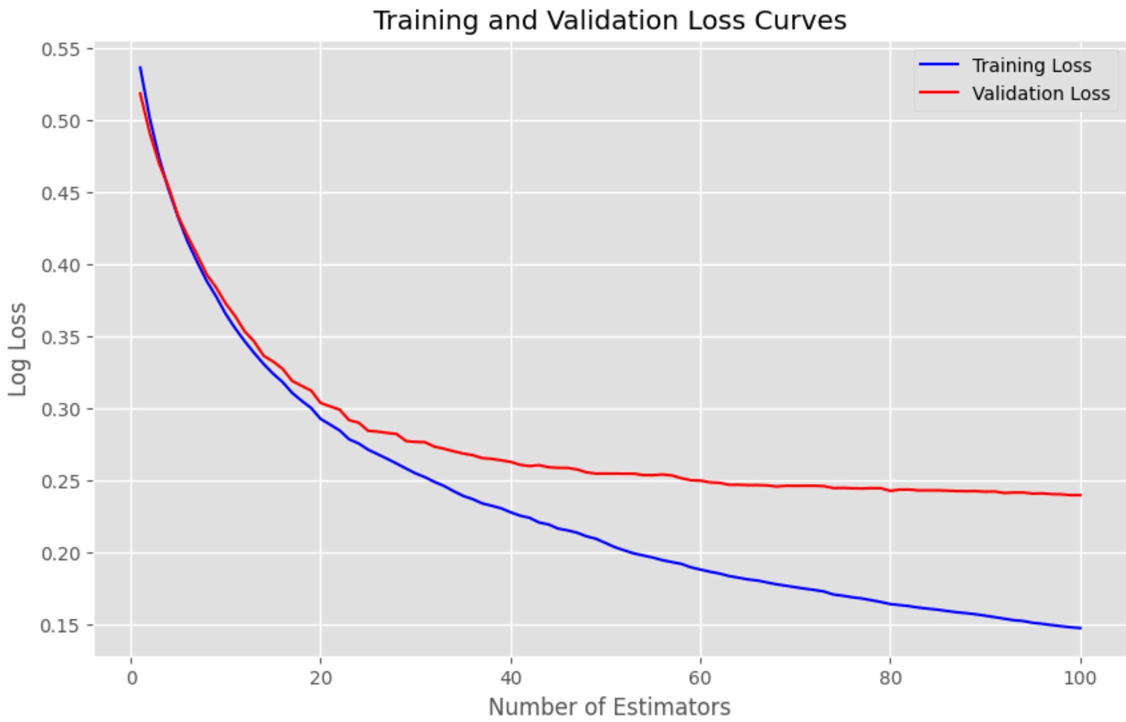
Gradient Boosting Classifier

```
import time
from sklearn.ensemble import GradientBoostingClassifier

# Initialize the Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Measure training time
start_time = time.time()

# Fit the model on the training data
gb_model.fit(X_train, y_train)

# Calculate training time
training_time4 = time.time() - start_time

# Predict on the training set
gb_train_pred = gb_model.predict(X_train)
gb_train_pred_proba = gb_model.predict_proba(X_train)

# Predict on the validation/test set
gb_test_pred = gb_model.predict(X_val)
gb_test_pred_proba = gb_model.predict_proba(X_val)

# Calculate accuracy for training and validation/test sets
gb_train_acc4 = accuracy_score(y_train, gb_train_pred)
gb_test_acc4 = accuracy_score(y_val, gb_test_pred)

# Calculate log loss (cross-entropy loss) for training and validation/test sets
gb_train_loss4 = log_loss(y_train, gb_train_pred_proba)
gb_test_loss4 = log_loss(y_val, gb_test_pred_proba)

# Calculate the number of trainable parameters
# This calculation is based on the number of estimators and the depth of each tree.
num_estimators = gb_model.n_estimators
# Each tree in GradientBoostingClassifier typically has several nodes,
# but counting exact parameters is non-trivial. You can approximate or ignore for practical purposes.

# Measure inference time
start_time_inference = time.time()
_ = gb_model.predict(X_val)  # Or use X_train if you want to measure on training set
inference_time4 = time.time() - start_time_inference

# Print results
print("GB Train accuracy: {:.2f}%".format(gb_train_acc4 * 100))
print("GB Validation accuracy: {:.2f}%".format(gb_test_acc4 * 100))
print("GB Train loss: {:.4f}".format(gb_train_loss4))
print("GB Test loss: {:.4f}".format(gb_test_loss4))
print("Training time: {:.4f} seconds".format(training_time4))
print("Inference time: {:.4f} seconds".format(inference_time4))
print("Number of estimators: {}".format(num_estimators))
```

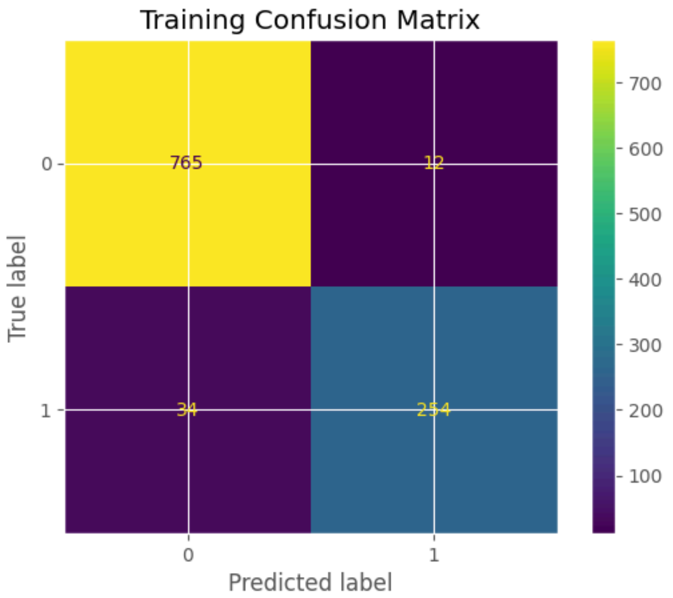
```
GB Train accuracy: 95.68%
GB Validation accuracy: 91.83%
GB Train loss: 0.1475
GB Test loss: 0.2398
Training time: 12.4493 seconds
Inference time: 0.0016 seconds
Number of estimators: 100
```

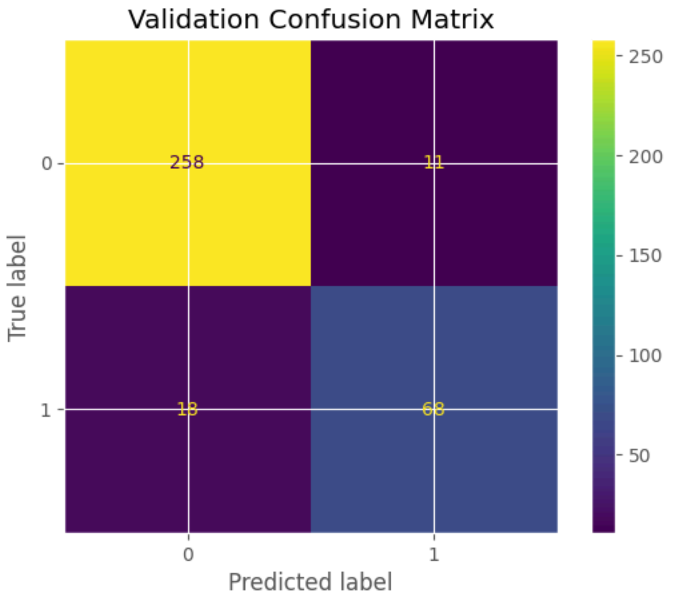
```
# Prepare lists to store losses
train_losses = []
val_losses = []

# Compute losses for each stage
for stage in gb_model.staged_predict_proba(X_train):
    train_losses.append(log_loss(y_train, stage))

for stage in gb_model.staged_predict_proba(X_val):
    val_losses.append(log_loss(y_val, stage))

# Plotting the loss curves
plt.figure(figsize=(10, 6))

# Plot training loss
plt.plot(range(1, len(train_losses) + 1), train_losses, label='Training Loss', color='blue')

# Plot validation loss
plt.plot(range(1, len(val_losses) + 1), val_losses, label='Validation Loss', color='red')

plt.xlabel('Number of Estimators')
plt.ylabel('Log Loss')
plt.title('Training and Validation Loss Curves')
```

```
plt.legend()
plt.grid(True)
plt.show()
```




```
train_conf_matrix = confusion_matrix(y_train, gb_train_pred)
ConfusionMatrixDisplay(train_conf_matrix).plot()
plt.title("Training Confusion Matrix")
plt.show()

val_conf_matrix = confusion_matrix(y_val, gb_test_pred)
ConfusionMatrixDisplay(val_conf_matrix).plot()
plt.title("Validation Confusion Matrix")
plt.show()
```





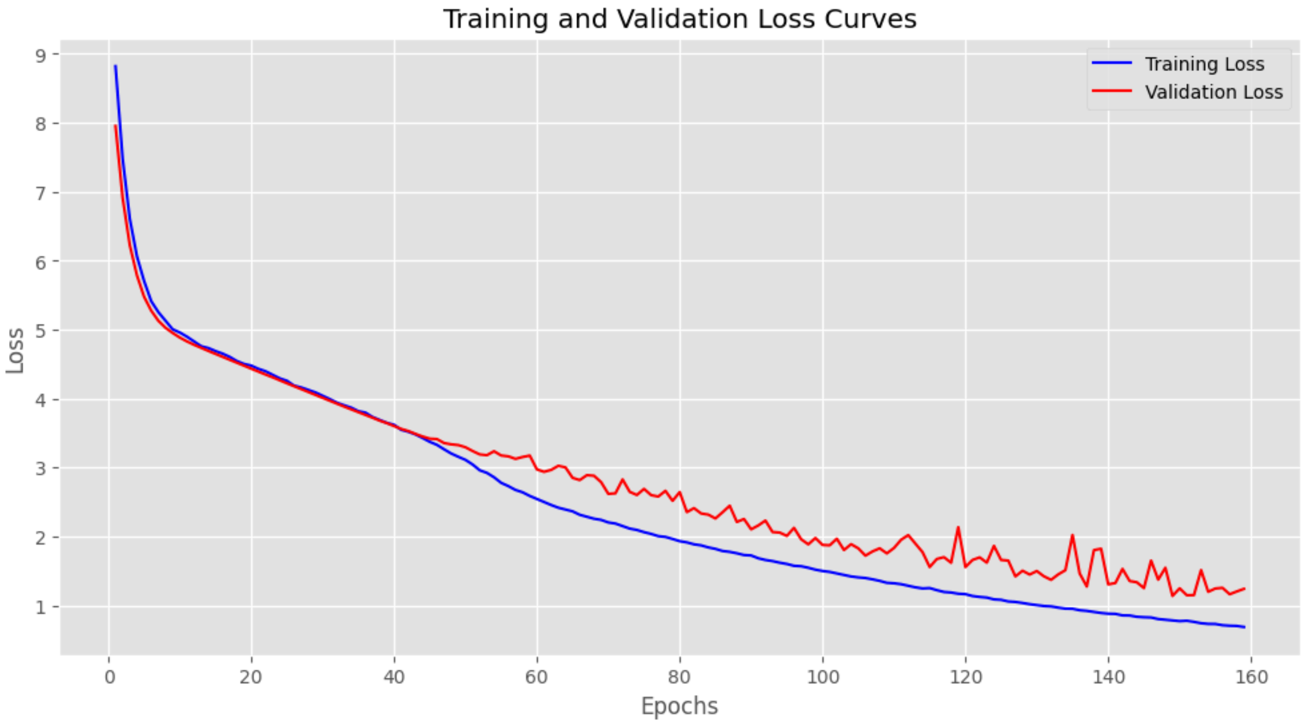
Neural Network + MHA

```
import time
import numpy as np
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, MultiHeadAttention, LayerNormalization, Input, Add, Flatten, Reshape
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score

# Define the input shape based on your TF-IDF vectors
input_shape = X_train.shape[1]

# Create a functional model to integrate MHA with increased regularization and dropout
input_layer = Input(shape=(input_shape,))
x = Dense(256, activation='relu', kernel_regularizer=l2(0.01))(input_layer)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(x)
x = Dropout(0.5)(x)
x = Dense(32, activation='relu', kernel_regularizer=l2(0.01))(x)
x = Dropout(0.5)(x)

x = Reshape((4, 8))(x)  # Reshape for MultiHeadAttention layer

# Add the Multi-Head Attention layer with fewer heads
mha_output = MultiHeadAttention(num_heads=2, key_dim=16)(x, x)
mha_output = LayerNormalization(epsilon=1e-6)(mha_output)
```

```
mha_output = Add()([x, mha_output])

# Flatten the output of the MHA layer before passing it to Dense layers
flattened_output = Flatten()(mha_output)

x = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(flattened_output)
x = Dropout(0.5)(x)
x = Dense(32, activation='relu', kernel_regularizer=l2(0.01))(x)
x = Dropout(0.5)(x)

output_layer = Dense(2, activation='softmax')(x)

# Define the model
model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model with a lower learning rate
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    optimizer=Adam(learning_rate=0.00005),
    metrics=['accuracy']
)

# Early stopping with reduced patience
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Measure training time
start_time = time.time()
history = model.fit(X_train, y_train, epochs=200, validation_split=0.2, callbacks=[early_stopping])
end_time = time.time()
training_time5 = end_time - start_time

# Measure inference time on the validation data
start_inference_time = time.time()
Val_predictions = model.predict(X_val)
end_inference_time = time.time()
inference_time5 = end_inference_time - start_inference_time

# Calculate training accuracy
train_predictions = model.predict(X_train)
train_pred_labels = np.argmax(train_predictions, axis=1)
train_accuracy5 = accuracy_score(y_train, train_pred_labels)

# Calculate validation accuracy
Val_pred_labels = np.argmax(Val_predictions, axis=1)
Val_accuracy5 = accuracy_score(y_val, Val_pred_labels)

# Get the number of trainable parameters
num_parameters = model.count_params()

# Get the final training and validation loss from history
training_loss5 = history.history['loss'][-1]
validation_loss5 = history.history['val_loss'][-1]

print(f'Training time: {training_time5:.2f} seconds')
print(f'Inference time on validation set: {inference_time5:.2f} seconds')
print(f'Number of trainable parameters: {num_parameters}')
print(f'Training loss: {training_loss5:.4f}')
print(f'Validation loss: {validation_loss5:.4f}')
print(f'Training accuracy: {train_accuracy5 * 100:.2f}%')
print(f'Validation accuracy: {Val_accuracy5 * 100:.2f}%')
```

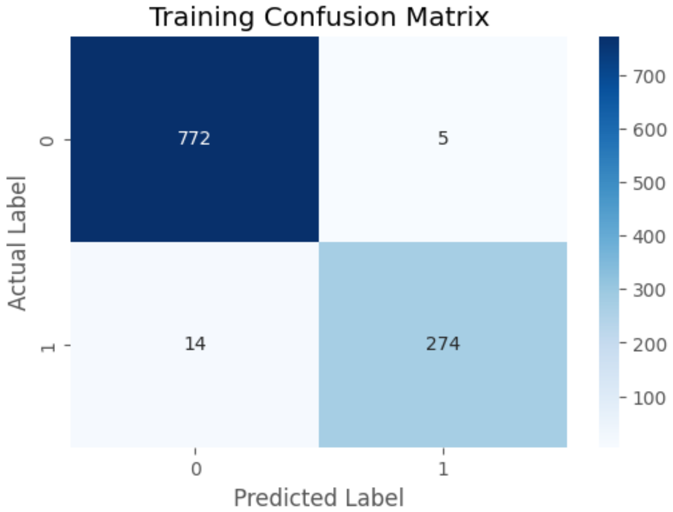
```
Epoch 147/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 450ms/step - accuracy: 0.9998 - loss: 0.8058 - val_accuracy: 0.8873 - val_loss: 1.3761
Epoch 148/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 455ms/step - accuracy: 0.9994 - loss: 0.7974 - val_accuracy: 0.8592 - val_loss: 1.5486
Epoch 149/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 451ms/step - accuracy: 0.9995 - loss: 0.7839 - val_accuracy: 0.9108 - val_loss: 1.1405
Epoch 150/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 448ms/step - accuracy: 0.9971 - loss: 0.7770 - val_accuracy: 0.9108 - val_loss: 1.2505
Epoch 151/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 455ms/step - accuracy: 0.9898 - loss: 0.7856 - val_accuracy: 0.9155 - val_loss: 1.1506
Epoch 152/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 431ms/step - accuracy: 0.9951 - loss: 0.7654 - val_accuracy: 0.8920 - val_loss: 1.1521
Epoch 153/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 452ms/step - accuracy: 0.9955 - loss: 0.7513 - val_accuracy: 0.8732 - val_loss: 1.5149
Epoch 154/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 470ms/step - accuracy: 0.9977 - loss: 0.7356 - val_accuracy: 0.9108 - val_loss: 1.2002
Epoch 155/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 442ms/step - accuracy: 0.9956 - loss: 0.7619 - val_accuracy: 0.9061 - val_loss: 1.2480
Epoch 156/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 21s 452ms/step - accuracy: 0.9979 - loss: 0.7161 - val_accuracy: 0.9014 - val_loss: 1.2575
Epoch 157/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 454ms/step - accuracy: 0.9975 - loss: 0.7103 - val_accuracy: 0.9014 - val_loss: 1.1649
Epoch 158/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 12s 453ms/step - accuracy: 0.9952 - loss: 0.7131 - val_accuracy: 0.9061 - val_loss: 1.2049
Epoch 159/200
27/27 ━━━━━━━━━━━━━━━━━━━━ 20s 450ms/step - accuracy: 0.9956 - loss: 0.6943 - val_accuracy: 0.9014 - val_loss: 1.2424
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step
Training time: 2749.52 seconds
Inference time on validation set: 1.30 seconds
Number of trainable parameters: 13564154
Training loss: 0.6900
Validation loss: 1.2424
Training accuracy: 98.22%
Validation accuracy: 90.42%
```

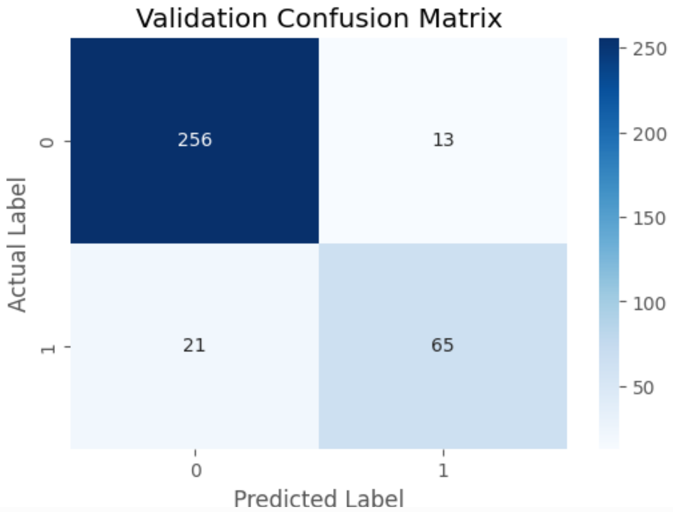
```
# Extract loss values
epochs = range(1, len(history.history['loss']) + 1)
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot training and validation loss curves
plt.figure(figsize=(12, 6))
plt.plot(epochs, train_loss, 'b-', label='Training Loss')
plt.plot(epochs, val_loss, 'r-', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Curves')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Training confusion matrix
train_conf_matrix = confusion_matrix(y_train, train_pred_labels)
```

```
# Validation confusion matrix
val_conf_matrix = confusion_matrix(y_val, Val_pred_labels)

# Function to plot confusion matrix
def plot_confusion_matrix(conf_matrix, title):
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.title(title)
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')
    plt.show()

# Plot confusion matrices
plot_confusion_matrix(train_conf_matrix, 'Training Confusion Matrix')
plot_confusion_matrix(val_conf_matrix, 'Validation Confusion Matrix')
```





Bert model

```
!pip install --upgrade transformers tensorflow
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.1)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.19.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
```

```
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.5.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->trans
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.1.
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow) (3
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow) (3
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard~=2.1
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflo
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorf
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.
```

```
pip install hf_xet
```

```
Requirement already satisfied: hf_xet in /usr/local/lib/python3.11/dist-packages (1.0.2)
```

```
import time
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.callbacks import EarlyStopping
from transformers import BertTokenizer, TFBertModel
import numpy as np
from sklearn.metrics import accuracy_score

# Assuming 'covid_hate' is your DataFrame and 'Text' is your column with raw text data
x_train_raw = covid_hate['Text'].tolist()
x_test_raw = covid_hate['Text'].tolist()

# Extract labels from your dataset
y_train = covid_hate['label'].values  # Replace 'label' with your actual label column name
y_test = covid_hate['label'].values  # Replace 'label' with your actual label column name

# Load BERT tokenizer and model
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
bert_model = TFBertModel.from_pretrained(model_name,from_pt=True)

# Tokenize and process text data in smaller batches to avoid high memory usage
def tokenize_texts(texts, tokenizer, max_length=128):
    return tokenizer(
        texts,
        max_length=max_length,
        padding='max_length',
        truncation=True,
        return_tensors='tf'
    )

# Use smaller batches to handle large datasets efficiently
def process_in_batches(texts, tokenizer, batch_size=32, max_length=128):
    embeddings = []
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]
        batch_tokens = tokenize_texts(batch_texts, tokenizer, max_length)
        batch_embeddings = get_bert_embeddings(batch_tokens, bert_model)
        embeddings.append(batch_embeddings)
    return tf.concat(embeddings, axis=0)

# Extract BERT embeddings
def get_bert_embeddings(tokens, model):
    outputs = model(input_ids=tokens['input_ids'], attention_mask=tokens['attention_mask'])
    return outputs.last_hidden_state[:, 0, :]  # Use [CLS] token representation
```

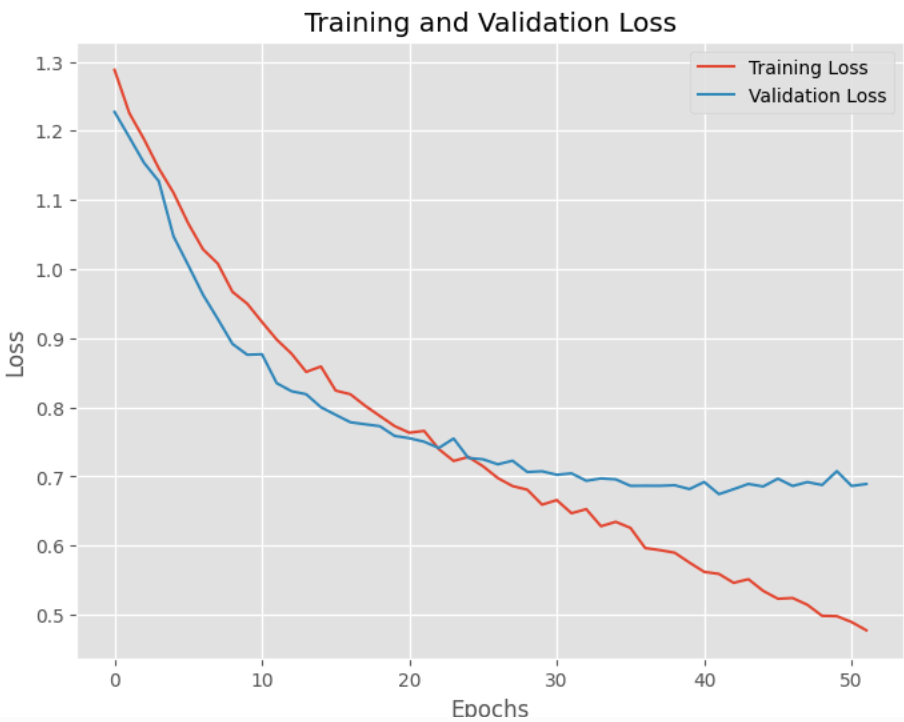
```
# Process data in batches
x_train_embeddings = process_in_batches(x_train_raw, tokenizer)
x_test_embeddings = process_in_batches(x_test_raw, tokenizer)

# Define MLP Model for Classification
input_shape = x_train_embeddings.shape[1]
input_layer = Input(shape=(input_shape,))
x = Dense(256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(input_layer)
x = Dropout(0.3)(x)
x = Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
x = Dropout(0.3)(x)
x = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
x = Dropout(0.3)(x)
output_layer = Dense(2, activation='softmax')(x)

# Define the model
mlp_model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
mlp_model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    optimizer=tf.keras.optimizers.Adam(0.0001),  # Lower learning rate
    metrics=['accuracy']
)

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Measure training time
start_time = time.time()
history = mlp_model.fit(x_train_embeddings, y_train, epochs=200, validation_split=0.2, callbacks=[early_stopping])
end_time = time.time()
training_time6 = end_time - start_time

# Measure inference time on the test data
start_inference_time = time.time()
test_loss, test_acc = mlp_model.evaluate(x_test_embeddings, y_test)
end_inference_time = time.time()
inference_time6 = end_inference_time - start_inference_time

# Manually predict labels for the test set (optional, but not necessary if using evaluate)
test_predictions = mlp_model.predict(x_test_embeddings)
test_pred_labels = np.argmax(test_predictions, axis=1)
test_accuracy_manual = accuracy_score(y_test, test_pred_labels)

# Calculate training accuracy
train_predictions = mlp_model.predict(x_train_embeddings)
train_pred_labels = np.argmax(train_predictions, axis=1)
train_accuracy6 = accuracy_score(y_train, train_pred_labels)

# Calculate validation accuracy (from history)
val_predictions = mlp_model.predict(x_train_embeddings[int(len(x_train_embeddings) * 0.8):])
val_pred_labels = np.argmax(val_predictions, axis=1)
y_val = y_train[int(len(y_train) * 0.8):]
val_accuracy6 = accuracy_score(y_val, val_pred_labels)

# Get the number of trainable parameters
num_parameters = mlp_model.count_params()

# Get the final training and validation loss from history
training_loss6 = history.history['loss'][-1]
validation_loss6 = history.history['val_loss'][-1]

# Print all the metrics
print(f'Training time: {training_time6:.2f} seconds')
print(f'Inference time on test set: {inference_time6:.2f} seconds')
print(f'Number of trainable parameters: {num_parameters}')
print(f'Training loss: {training_loss6:.4f}')
print(f'Validation loss: {validation_loss6:.4f}')
print(f'Training accuracy: {train_accuracy6 * 100:.2f}%')
print(f'Validation accuracy: {val_accuracy6 * 100:.2f}%')
print(f'Test accuracy (from evaluate): {test_acc * 100:.2f}%')
print(f'Test accuracy (manual calculation): {test_accuracy_manual * 100:.2f}%')
```

pytorch_model.bin: 100% 440M/440M [00:16<00:00, 28.0MB/s]

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.decoder.weight',
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initia
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions witl
Epoch 1/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.6870 - loss: 1.3065 - val_accuracy: 0.7268 - val_loss: 1.2273
Epoch 2/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.7321 - loss: 1.2361 - val_accuracy: 0.7296 - val_loss: 1.1908
Epoch 3/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7357 - loss: 1.2069 - val_accuracy: 0.7268 - val_loss: 1.1537
Epoch 4/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7611 - loss: 1.1396 - val_accuracy: 0.7408 - val_loss: 1.1271
Epoch 5/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7389 - loss: 1.1381 - val_accuracy: 0.7775 - val_loss: 1.0473
Epoch 6/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7702 - loss: 1.0783 - val_accuracy: 0.8535 - val_loss: 1.0056
Epoch 7/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7853 - loss: 1.0353 - val_accuracy: 0.8535 - val_loss: 0.9630
Epoch 8/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7938 - loss: 1.0157 - val_accuracy: 0.8648 - val_loss: 0.9280
Epoch 9/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8212 - loss: 0.9843 - val_accuracy: 0.8620 - val_loss: 0.8919
Epoch 10/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8151 - loss: 0.9435 - val_accuracy: 0.8761 - val_loss: 0.8762
Epoch 11/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8528 - loss: 0.9097 - val_accuracy: 0.8563 - val_loss: 0.8769
Epoch 12/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8367 - loss: 0.8963 - val_accuracy: 0.8789 - val_loss: 0.8351
Epoch 13/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8606 - loss: 0.8723 - val_accuracy: 0.8817 - val_loss: 0.8235
Epoch 14/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8674 - loss: 0.8409 - val_accuracy: 0.8563 - val_loss: 0.8191
Epoch 15/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8530 - loss: 0.8754 - val_accuracy: 0.8817 - val_loss: 0.8003
Epoch 16/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8463 - loss: 0.8569 - val_accuracy: 0.8789 - val_loss: 0.7893
Epoch 17/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8666 - loss: 0.8097 - val_accuracy: 0.8873 - val_loss: 0.7785
Epoch 18/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8782 - loss: 0.7931 - val_accuracy: 0.8761 - val_loss: 0.7756
Epoch 19/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8727 - loss: 0.7902 - val_accuracy: 0.8592 - val_loss: 0.7728
Epoch 20/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8879 - loss: 0.7794 - val_accuracy: 0.8901 - val_loss: 0.7588
Epoch 21/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8851 - loss: 0.7573 - val_accuracy: 0.8873 - val_loss: 0.7555
Epoch 22/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8738 - loss: 0.7668 - val_accuracy: 0.8732 - val_loss: 0.7504
Epoch 23/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8902 - loss: 0.7411 - val_accuracy: 0.8817 - val_loss: 0.7413
Epoch 24/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8893 - loss: 0.7312 - val_accuracy: 0.8507 - val_loss: 0.7550
Epoch 25/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9027 - loss: 0.6994 - val_accuracy: 0.8845 - val_loss: 0.7269
Epoch 26/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9003 - loss: 0.7091 - val_accuracy: 0.8901 - val_loss: 0.7250
Epoch 27/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9095 - loss: 0.6865 - val_accuracy: 0.8845 - val_loss: 0.7177
Epoch 28/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9132 - loss: 0.6800 - val_accuracy: 0.8620 - val_loss: 0.7229
Epoch 29/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9098 - loss: 0.6686 - val_accuracy: 0.8958 - val_loss: 0.7066
Epoch 30/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9216 - loss: 0.6515 - val_accuracy: 0.8901 - val_loss: 0.7077
Epoch 31/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9063 - loss: 0.6698 - val_accuracy: 0.8958 - val_loss: 0.7026
Epoch 32/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9145 - loss: 0.6547 - val_accuracy: 0.8761 - val_loss: 0.7046
Epoch 33/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9302 - loss: 0.6311 - val_accuracy: 0.8901 - val_loss: 0.6939
Epoch 34/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9348 - loss: 0.6294 - val_accuracy: 0.8845 - val_loss: 0.6973
Epoch 35/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9180 - loss: 0.6364 - val_accuracy: 0.8732 - val_loss: 0.6960
Epoch 36/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9187 - loss: 0.6270 - val_accuracy: 0.8986 - val_loss: 0.6866
Epoch 37/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9380 - loss: 0.5797 - val_accuracy: 0.8930 - val_loss: 0.6867
Epoch 38/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9337 - loss: 0.5890 - val_accuracy: 0.8930 - val_loss: 0.6866
Epoch 39/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9302 - loss: 0.6091 - val_accuracy: 0.8958 - val_loss: 0.6874
Epoch 40/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9427 - loss: 0.5673 - val_accuracy: 0.8958 - val_loss: 0.6819
Epoch 41/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9366 - loss: 0.5830 - val_accuracy: 0.8901 - val_loss: 0.6921
Epoch 42/200
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9395 - loss: 0.5588 - val_accuracy: 0.9014 - val_loss: 0.6746
Epoch 43/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9532 - loss: 0.5453 - val_accuracy: 0.8958 - val_loss: 0.6818
Epoch 44/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9376 - loss: 0.5677 - val_accuracy: 0.8958 - val_loss: 0.6893
Epoch 45/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9460 - loss: 0.5481 - val_accuracy: 0.8986 - val_loss: 0.6856
Epoch 46/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9557 - loss: 0.5186 - val_accuracy: 0.8986 - val_loss: 0.6970
Epoch 47/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9374 - loss: 0.5246 - val_accuracy: 0.8986 - val_loss: 0.6864
Epoch 48/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9586 - loss: 0.5177 - val_accuracy: 0.8732 - val_loss: 0.6921
Epoch 49/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9584 - loss: 0.4972 - val_accuracy: 0.8958 - val_loss: 0.6878
Epoch 50/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9528 - loss: 0.5032 - val_accuracy: 0.8958 - val_loss: 0.7078
Epoch 51/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9488 - loss: 0.5005 - val_accuracy: 0.8789 - val_loss: 0.6864
Epoch 52/200
45/45 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9663 - loss: 0.4766 - val_accuracy: 0.8873 - val_loss: 0.6893
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9665 - loss: 0.5071
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Training time: 32.70 seconds
Inference time on test set: 0.23 seconds
Number of trainable parameters: 240162
Training loss: 0.4777
Validation loss: 0.6893
Training accuracy: 95.72%
Validation accuracy: 90.14%
Test accuracy (from evaluate): 95.72%
Test accuracy (manual calculation): 95.72%
```

```python
# Plot training & validation loss values
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```




```python
# Predict on the test set
y_test_pred = np.argmax(mlp_model.predict(x_test_embeddings), axis=1)

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)

# Plot confusion matrix using seaborn heatmap
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```