Imorting Libraries

```
import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import nltk
import re
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem import WordNetLemmatizer
import itertools
from wordcloud import WordCloud
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from keras.models import Sequential,Model
from keras.layers import Dense,LSTM, SpatialDropout1D, Embedding
from tensorflow.keras import utils
from tensorflow.keras.utils import to_categorical
from joblib import dump, load
```

Reading the dataset

```
text = []
clas = []
df = pd.read_csv('https://raw.githubusercontent.com/shakil1819/NLTK-LSTM-Based-Hate-Speech-Detection/main/Dataset/labeled_data.csv')
text = df['tweet'].tolist()
clas = df['class'].tolist()
df.head()
```

```
   Unnamed: 0  count  hate_speech  offensive_language  neither  class  \
0           0      3            0                   0        3      2

1           1      3            0                   3        0      1
```

```
2           2       3           0                   3        0       1

3           3       3           0                   2        1       1

4           4       6           0                   6        0       1
```

```
                                               tweet
0  !!! RT @mayasolovely: As a woman you shouldn't t...
1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4  !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
```

creating a new dataframe for easy text processing

```
df = pd.DataFrame({'tweet': text, 'class': clas})
```

Finding if there is any missing data

```
print(df.isnull().sum())
```

```
tweet    0
class    0
dtype: int64
```

Converting the data into lower case.

```
df['tweet'] = df['tweet'].apply(lambda x:x.lower())
```

removing punctuations

```
punctuation_signs = list("?:!.,;")
df['tweet'] = df['tweet']

for punct_sign in punctuation_signs:
    df['tweet'] = df['tweet'].str.replace(punct_sign, '')
```

```
<ipython-input-26-b7a77ccdace9>:5: FutureWarning: The default value of
regex will change from True to False in a future version. In addition,
```

```
single character regular expressions will *not* be treated as literal
strings when regex=True.
  df['tweet'] = df['tweet'].str.replace(punct_sign, '')
```

Removing '\n' and '\t', extra spaces, quoting text, and progressive pronouns.

```
df['tweet'] = df['tweet'].apply(lambda x: x.replace('\n', ' '))
df['tweet'] = df['tweet'].apply(lambda x: x.replace('\t', ' '))
df['tweet'] = df['tweet'].str.replace("    ", " ")
df['tweet'] = df['tweet'].str.replace('"', '')
df['tweet'] = df['tweet'].str.replace("'s", "")
```

removing stop-words

```
nltk.download('stopwords')
stop_words = list(stopwords.words('english'))
for stop_word in stop_words:
    regex_stopword = r"\b" + stop_word + r"\b"
    df['tweet'] = df['tweet'].str.replace(regex_stopword, '')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
<ipython-input-28-969333b8111c>:5: FutureWarning: The default value of
regex will change from True to False in a future version.
  df['tweet'] = df['tweet'].str.replace(regex_stopword, '')
```

Using Bag of Words approach for final data Preparation.¶

```
cv = CountVectorizer(max_features = 75)
X = cv.fit_transform(df['tweet']).toarray()
y = df['class']
```

Splitting the Data using Stratified split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, stratify=y, random_state = 42)

def plot_confusion_matrix(cm, classes,
                          normalize=False,
```

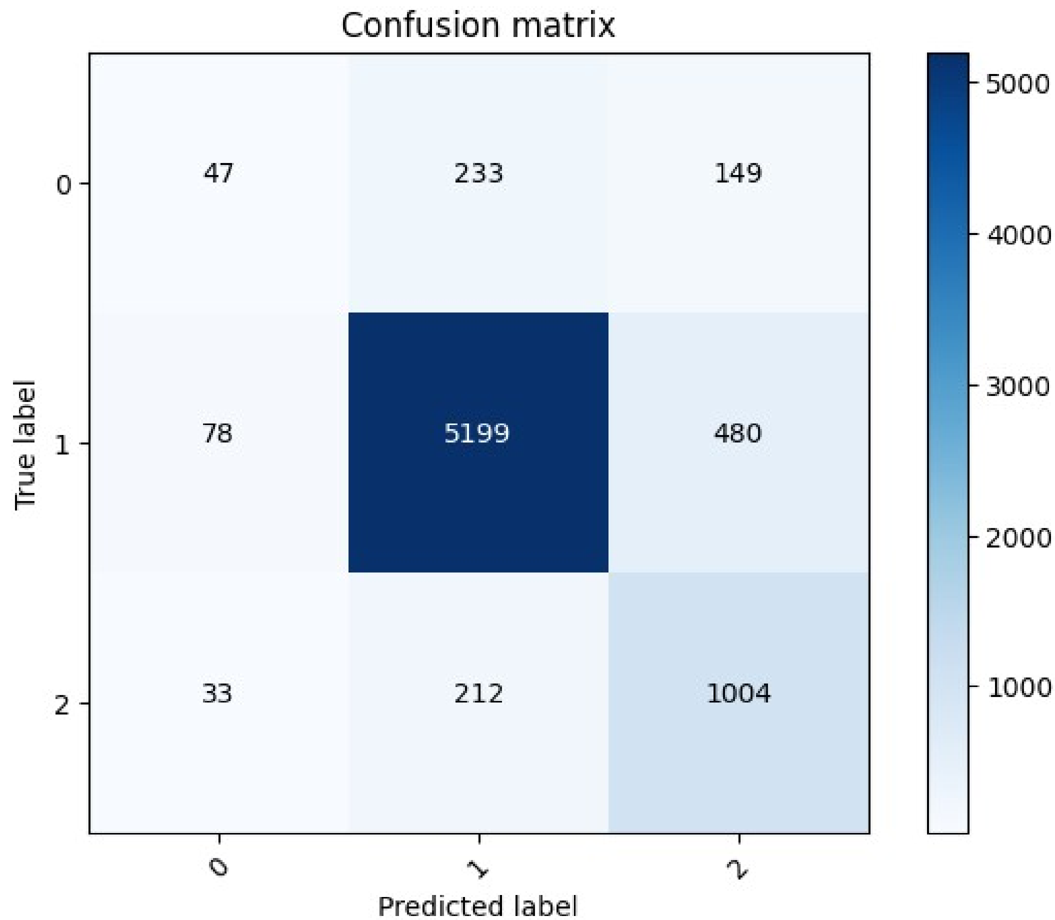
```
                          title='Confusion matrix',
                              cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Using Random Forest Classifier as the Model and printing evaluating it using confusion matrix

```
clf = RandomForestClassifier(n_estimators=10)
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy is: ",accuracy)
CM = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(CM, classes = range(3))
dump(clf, 'rf.joblib')
```

```
accuracy is:  0.8406186953597848
```

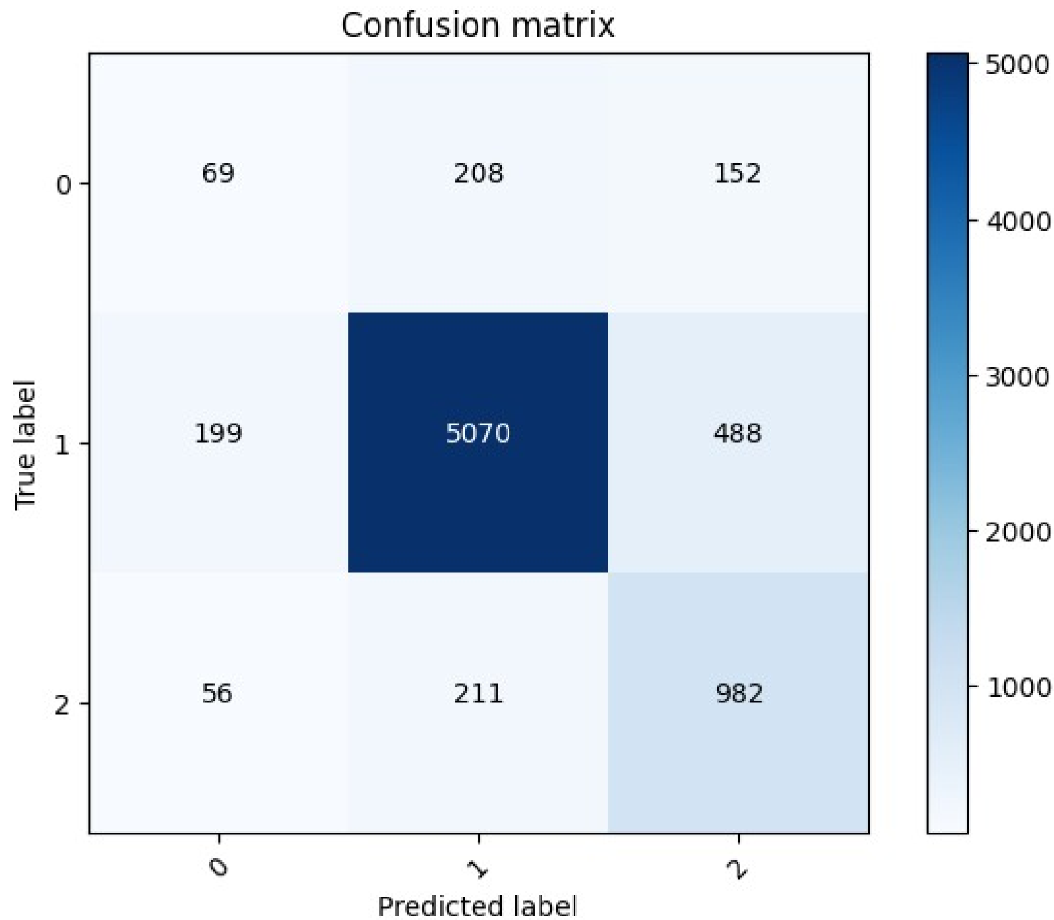
```
['rf.joblib']
```



Using Decision tree as the Model and printing evaluating it using confusion matrix

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy is: ",accuracy)
CM = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(CM, classes = range(3))
dump(clf, 'decision.joblib')
```

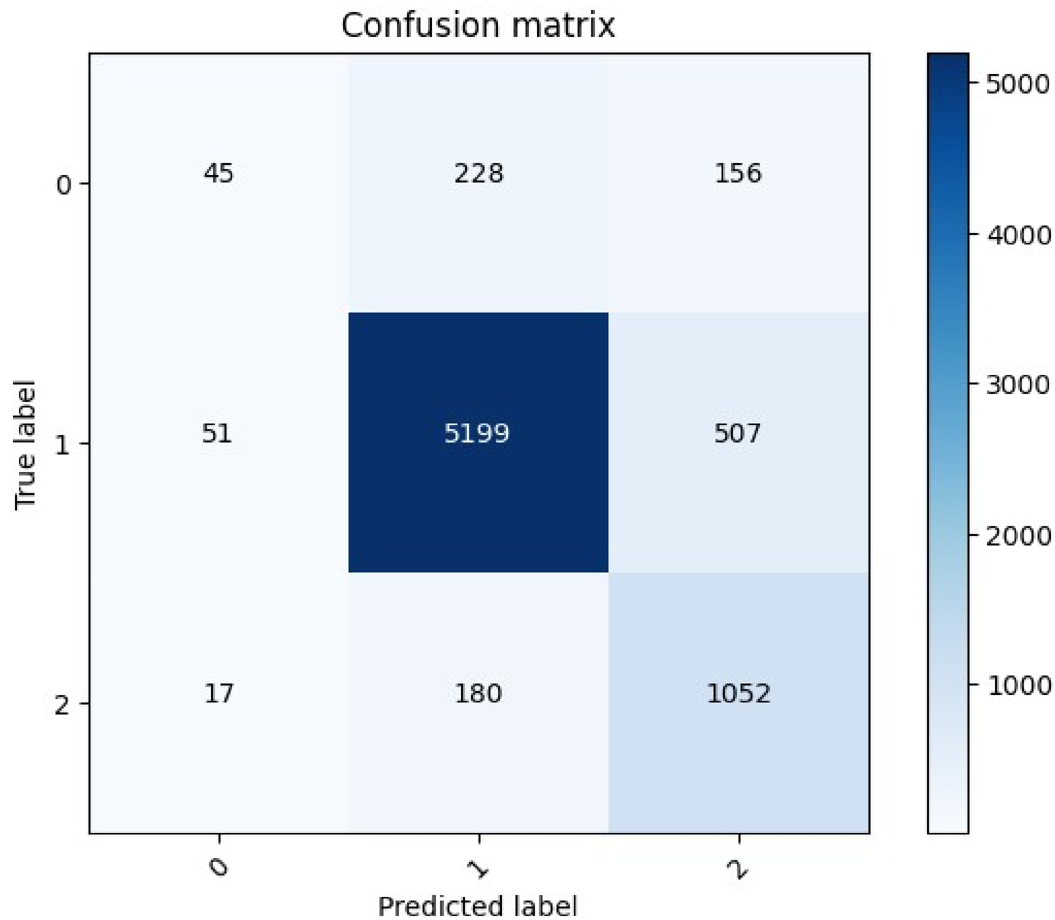
```
accuracy is:  0.8232683254875588

['decision.joblib']
```



Using AdaBoost Classifier as the Model and printing evaluating it using confusion matrix

```
clf = AdaBoostClassifier(n_estimators=100)
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy is: ",accuracy)
CM = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(CM, classes = range(3))
dump(clf, 'ada.joblib')
```

```
accuracy is:  0.8468056489576328

['ada.joblib']
```



Converting the labels into categorical format

```
y_train=to_categorical(y_train, num_classes = 3, dtype='float32')
y_test=to_categorical(y_test, num_classes = 3, dtype='float32')
```

Creating and Training an LSTM Model

```
model = Sequential()
model.add(Embedding(232337, 100, input_length=X_train.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(20, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

epochs = 50
batch_size = 64
```

```
history = model.fit(X_train, y_train,validation_data =
(X_test,y_test), epochs=epochs, batch_size=batch_size)
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
```

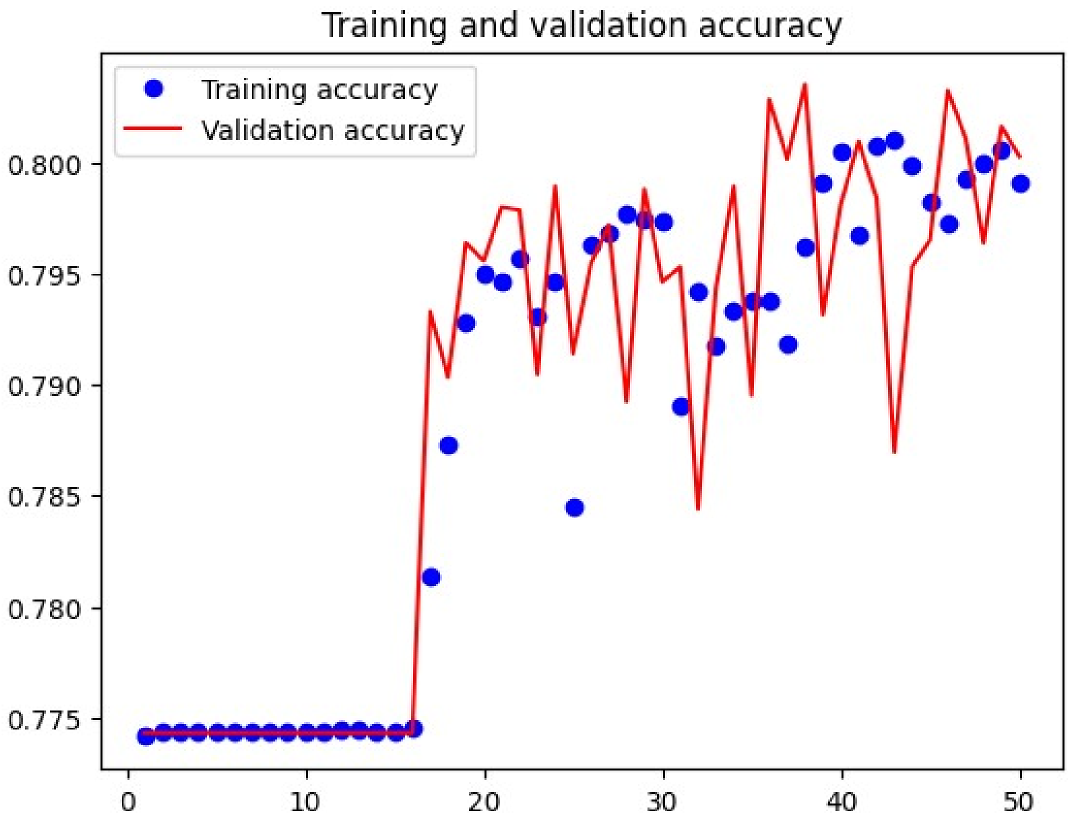
```
Epoch 1/50
272/272 [==============================] - 108s 370ms/step - loss:
0.4182 - accuracy: 0.7742 - val_loss: 0.4024 - val_accuracy: 0.7743
Epoch 2/50
272/272 [==============================] - 98s 361ms/step - loss:
0.4025 - accuracy: 0.7743 - val_loss: 0.4019 - val_accuracy: 0.7743
Epoch 3/50
272/272 [==============================] - 98s 361ms/step - loss:
0.4021 - accuracy: 0.7743 - val_loss: 0.4012 - val_accuracy: 0.7743
Epoch 4/50
272/272 [==============================] - 104s 381ms/step - loss:
0.4015 - accuracy: 0.7743 - val_loss: 0.4009 - val_accuracy: 0.7743
Epoch 5/50
272/272 [==============================] - 96s 354ms/step - loss:
0.4013 - accuracy: 0.7743 - val_loss: 0.4008 - val_accuracy: 0.7743
Epoch 6/50
272/272 [==============================] - 98s 360ms/step - loss:
0.4012 - accuracy: 0.7743 - val_loss: 0.4006 - val_accuracy: 0.7743
Epoch 7/50
272/272 [==============================] - 97s 356ms/step - loss:
0.4008 - accuracy: 0.7743 - val_loss: 0.4006 - val_accuracy: 0.7743
Epoch 8/50
272/272 [==============================] - 99s 362ms/step - loss:
0.4008 - accuracy: 0.7743 - val_loss: 0.4004 - val_accuracy: 0.7743
Epoch 9/50
272/272 [==============================] - 98s 361ms/step - loss:
0.4007 - accuracy: 0.7743 - val_loss: 0.4001 - val_accuracy: 0.7743
Epoch 10/50
272/272 [==============================] - 101s 370ms/step - loss:
0.3999 - accuracy: 0.7743 - val_loss: 0.3978 - val_accuracy: 0.7743
Epoch 11/50
272/272 [==============================] - 98s 359ms/step - loss:
0.3947 - accuracy: 0.7743 - val_loss: 0.3885 - val_accuracy: 0.7743
Epoch 12/50
272/272 [==============================] - 99s 365ms/step - loss:
0.3907 - accuracy: 0.7744 - val_loss: 0.3867 - val_accuracy: 0.7743
Epoch 13/50
272/272 [==============================] - 97s 358ms/step - loss:
0.3883 - accuracy: 0.7744 - val_loss: 0.3955 - val_accuracy: 0.7743
Epoch 14/50
272/272 [==============================] - 98s 361ms/step - loss:
0.3930 - accuracy: 0.7743 - val_loss: 0.3886 - val_accuracy: 0.7743
Epoch 15/50
```

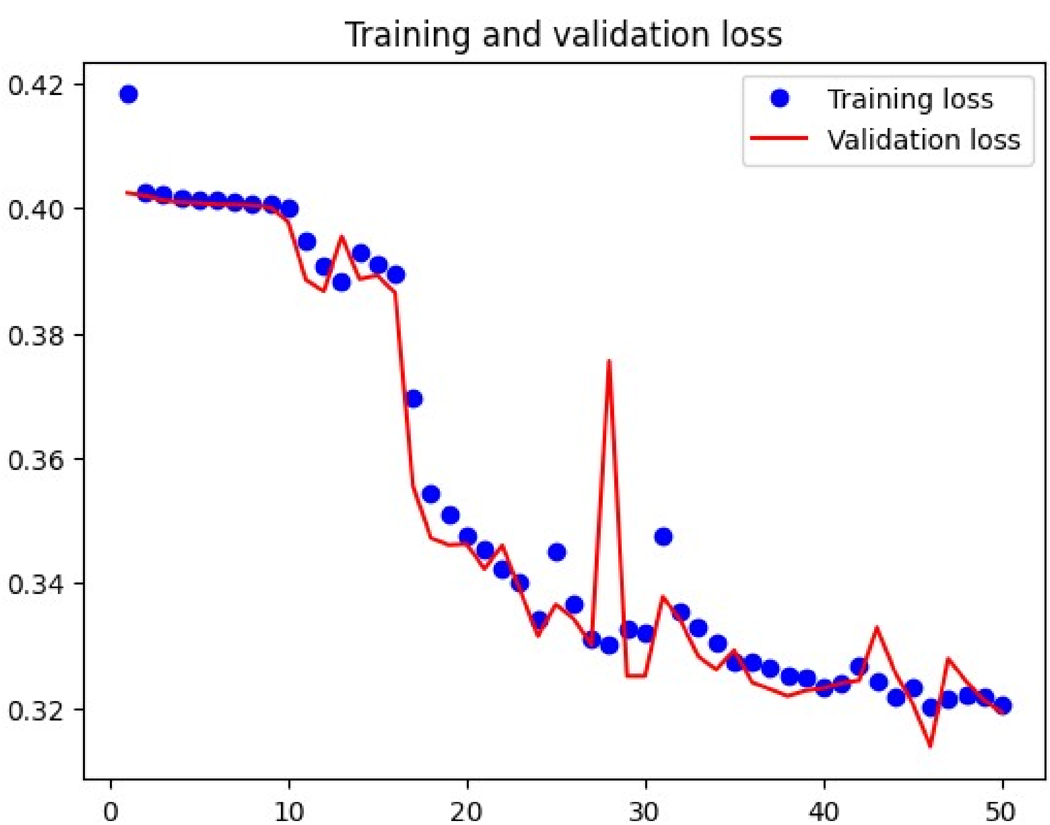
```
272/272 [==============================] - 98s 362ms/step - loss:
0.3912 - accuracy: 0.7743 - val_loss: 0.3892 - val_accuracy: 0.7743
Epoch 16/50
272/272 [==============================] - 97s 358ms/step - loss:
0.3894 - accuracy: 0.7746 - val_loss: 0.3864 - val_accuracy: 0.7743
Epoch 17/50
272/272 [==============================] - 97s 358ms/step - loss:
0.3696 - accuracy: 0.7814 - val_loss: 0.3556 - val_accuracy: 0.7933
Epoch 18/50
272/272 [==============================] - 97s 357ms/step - loss:
0.3544 - accuracy: 0.7873 - val_loss: 0.3472 - val_accuracy: 0.7903
Epoch 19/50
272/272 [==============================] - 97s 356ms/step - loss:
0.3509 - accuracy: 0.7928 - val_loss: 0.3461 - val_accuracy: 0.7964
Epoch 20/50
272/272 [==============================] - 96s 355ms/step - loss:
0.3474 - accuracy: 0.7950 - val_loss: 0.3463 - val_accuracy: 0.7956
Epoch 21/50
272/272 [==============================] - 96s 354ms/step - loss:
0.3455 - accuracy: 0.7947 - val_loss: 0.3423 - val_accuracy: 0.7980
Epoch 22/50
272/272 [==============================] - 97s 356ms/step - loss:
0.3422 - accuracy: 0.7957 - val_loss: 0.3460 - val_accuracy: 0.7978
Epoch 23/50
272/272 [==============================] - 96s 352ms/step - loss:
0.3402 - accuracy: 0.7931 - val_loss: 0.3389 - val_accuracy: 0.7905
Epoch 24/50
272/272 [==============================] - 95s 351ms/step - loss:
0.3341 - accuracy: 0.7946 - val_loss: 0.3315 - val_accuracy: 0.7989
Epoch 25/50
272/272 [==============================] - 95s 350ms/step - loss:
0.3450 - accuracy: 0.7845 - val_loss: 0.3366 - val_accuracy: 0.7914
Epoch 26/50
272/272 [==============================] - 94s 346ms/step - loss:
0.3367 - accuracy: 0.7963 - val_loss: 0.3343 - val_accuracy: 0.7954
Epoch 27/50
272/272 [==============================] - 96s 350ms/step - loss:
0.3310 - accuracy: 0.7968 - val_loss: 0.3301 - val_accuracy: 0.7972
Epoch 28/50
272/272 [==============================] - 96s 352ms/step - loss:
0.3300 - accuracy: 0.7977 - val_loss: 0.3755 - val_accuracy: 0.7892
Epoch 29/50
272/272 [==============================] - 96s 352ms/step - loss:
0.3326 - accuracy: 0.7974 - val_loss: 0.3251 - val_accuracy: 0.7988
Epoch 30/50
272/272 [==============================] - 96s 352ms/step - loss:
0.3320 - accuracy: 0.7973 - val_loss: 0.3251 - val_accuracy: 0.7946
Epoch 31/50
272/272 [==============================] - 97s 355ms/step - loss:
```

```
0.3474 - accuracy: 0.7891 - val_loss: 0.3378 - val_accuracy: 0.7953
Epoch 32/50
272/272 [==============================] - 96s 354ms/step - loss:
0.3354 - accuracy: 0.7942 - val_loss: 0.3340 - val_accuracy: 0.7844
Epoch 33/50
272/272 [==============================] - 96s 351ms/step - loss:
0.3329 - accuracy: 0.7918 - val_loss: 0.3282 - val_accuracy: 0.7944
Epoch 34/50
272/272 [==============================] - 98s 360ms/step - loss:
0.3304 - accuracy: 0.7933 - val_loss: 0.3262 - val_accuracy: 0.7989
Epoch 35/50
272/272 [==============================] - 98s 360ms/step - loss:
0.3273 - accuracy: 0.7938 - val_loss: 0.3293 - val_accuracy: 0.7895
Epoch 36/50
272/272 [==============================] - 97s 356ms/step - loss:
0.3274 - accuracy: 0.7938 - val_loss: 0.3241 - val_accuracy: 0.8028
Epoch 37/50
272/272 [==============================] - 96s 351ms/step - loss:
0.3264 - accuracy: 0.7918 - val_loss: 0.3230 - val_accuracy: 0.8001
Epoch 38/50
272/272 [==============================] - 95s 350ms/step - loss:
0.3253 - accuracy: 0.7962 - val_loss: 0.3219 - val_accuracy: 0.8035
Epoch 39/50
272/272 [==============================] - 96s 354ms/step - loss:
0.3248 - accuracy: 0.7991 - val_loss: 0.3228 - val_accuracy: 0.7931
Epoch 40/50
272/272 [==============================] - 96s 355ms/step - loss:
0.3234 - accuracy: 0.8005 - val_loss: 0.3232 - val_accuracy: 0.7981
Epoch 41/50
272/272 [==============================] - 96s 355ms/step - loss:
0.3239 - accuracy: 0.7967 - val_loss: 0.3240 - val_accuracy: 0.8009
Epoch 42/50
272/272 [==============================] - 95s 349ms/step - loss:
0.3266 - accuracy: 0.8008 - val_loss: 0.3244 - val_accuracy: 0.7984
Epoch 43/50
272/272 [==============================] - 95s 348ms/step - loss:
0.3243 - accuracy: 0.8010 - val_loss: 0.3329 - val_accuracy: 0.7870
Epoch 44/50
272/272 [==============================] - 95s 348ms/step - loss:
0.3217 - accuracy: 0.7999 - val_loss: 0.3258 - val_accuracy: 0.7953
Epoch 45/50
272/272 [==============================] - 96s 354ms/step - loss:
0.3232 - accuracy: 0.7982 - val_loss: 0.3208 - val_accuracy: 0.7965
Epoch 46/50
272/272 [==============================] - 96s 353ms/step - loss:
0.3203 - accuracy: 0.7972 - val_loss: 0.3139 - val_accuracy: 0.8032
Epoch 47/50
272/272 [==============================] - 96s 354ms/step - loss:
0.3215 - accuracy: 0.7992 - val_loss: 0.3279 - val_accuracy: 0.8011
```

```
Epoch 48/50
272/272 [==============================] - 96s 354ms/step - loss:
0.3221 - accuracy: 0.8000 - val_loss: 0.3243 - val_accuracy: 0.7964
Epoch 49/50
272/272 [==============================] - 96s 352ms/step - loss:
0.3217 - accuracy: 0.8006 - val_loss: 0.3213 - val_accuracy: 0.8016
Epoch 50/50
272/272 [==============================] - 96s 352ms/step - loss:
0.3207 - accuracy: 0.7991 - val_loss: 0.3193 - val_accuracy: 0.8003
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training accuracy
Validation accuracy
0.800
0.795
0.790
0.785
0.780
0.775
0
10
20
30
40
50


Training and validation loss
Training loss
Validation loss
0.42
0.40
0.38
0.36
0.34
0.32
0
10
20
30
40
50

Saving the LSTM Model

```
model.save('lstm.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
from sklearn.metrics import classification_report, accuracy_score,
precision_recall_fscore_support

# Get true labels
y_true = y_test

# Get LSTM predictions
lstm_predictions = (model.predict(X_test) > 0.5).astype("int32")

# Calculate accuracy
lstm_accuracy = accuracy_score(y_true, lstm_predictions)

# Get classification report
lstm_report = classification_report(y_true, lstm_predictions)

# Print accuracy
print("LSTM Accuracy:", lstm_accuracy)

# Print classification report
print(lstm_report)

# Get average precision, recall and F1 score
lstm_precision, lstm_recall, lstm_f1, _ =
precision_recall_fscore_support(y_true, lstm_predictions,
average='macro')

print("Average LSTM Precision:", lstm_precision)
print("Average LSTM Recall:", lstm_recall)
print("Average LSTM F1:", lstm_f1)
```

```
233/233 [==============================] - 6s 27ms/step
LSTM Accuracy: 0.7952925353059852
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       429
           1       0.86      0.92      0.89      5757
           2       0.54      0.52      0.53      1249

   micro avg       0.81      0.80      0.80      7435
   macro avg       0.47      0.48      0.47      7435
```

```
weighted avg       0.76      0.80      0.78      7435
 samples avg       0.80      0.80      0.80      7435

Average LSTM Precision: 0.4656287552307448
Average LSTM Recall: 0.4769487062060047
Average LSTM F1: 0.47094199848472096

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are 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, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in samples with no predicted labels. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are 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, msg_start, len(result))
```