In [ ]:
```
# This Python 3 environment comes with many helpful analytics libraries i
# It is defined by the kaggle/python Docker image: https://github.com/kag
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) wil

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) th
# You can also write temporary files to /kaggle/temp/, but they won't be
```

```
/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.csv
/kaggle/input/notebook/sentiment-imdb-lstm.ipynb
```

In [ ]:

In [ ]:
```
data = pd.read_csv("/kaggle/input/imdb-dataset-of-50k-movie-reviews/IMDB
```

In [ ]:
```
data.head()
```

Out[ ]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

In [ ]:
```
data.isnull().sum()
```

Out[ ]:
```
review       0
sentiment    0
dtype: int64
```

In [ ]:
```
x = data.drop("sentiment",axis=1)
```

In [ ]:
```
y = data.drop("review",axis=1)
```

In [ ]:
```
x.shape
```

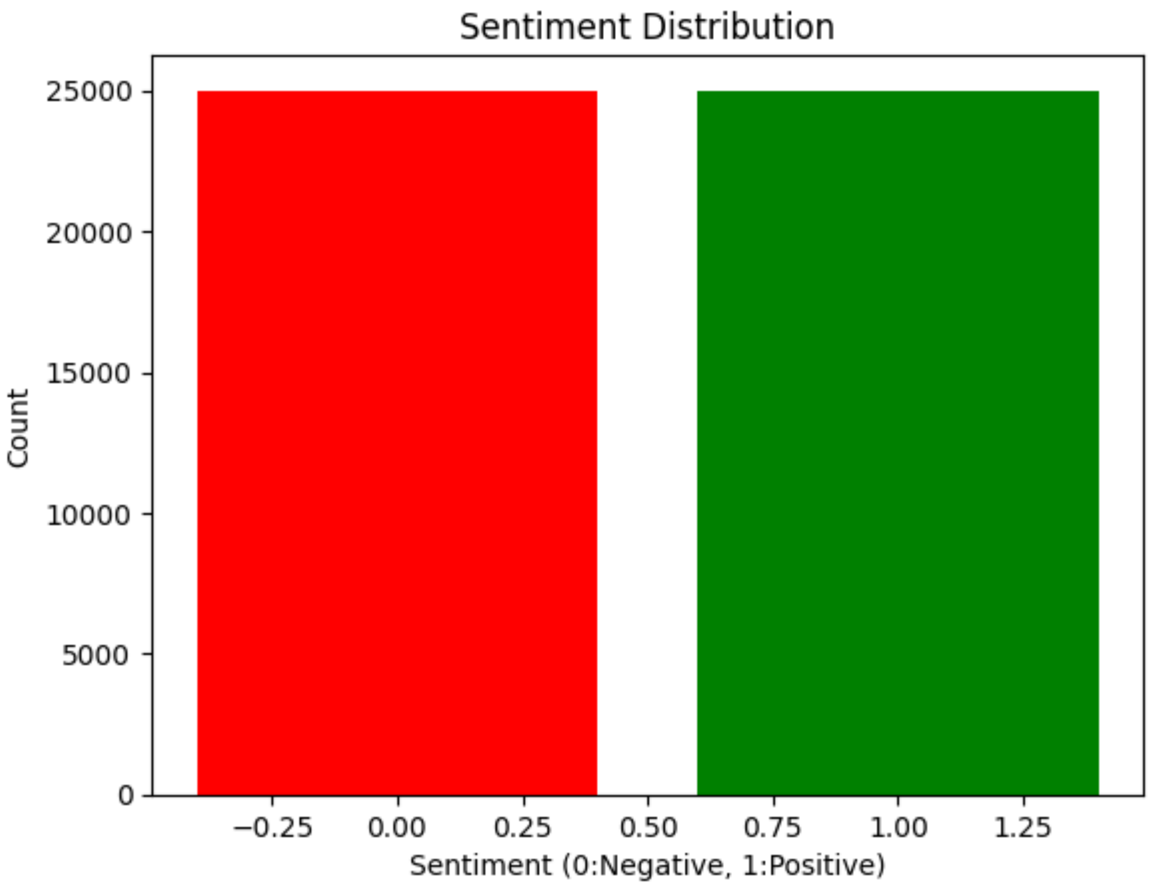
Out[ ]:
```
(50000, 1)
```

In [ ]:
```
y.shape
```

Out[ ]:
```
(50000, 1)
```

In [ ]:
```
# Mapping function
def convert_sentiment(sentiment):
    if sentiment.lower() == 'positive':
        return 1
    elif sentiment.lower() == 'negative':
        return 0
    else:
        return None

y['sentiment'] = y['sentiment'].apply(convert_sentiment)
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.bar(y['sentiment'].value_counts().index, y['sentiment'].value_counts(
plt.xlabel('Sentiment (0:Negative, 1:Positive)')
plt.ylabel('Count')
plt.title('Sentiment Distribution')
plt.show()
```



In [ ]:
```
# No class imbalance (no need for undersmapling or oversampling)
```

In [ ]:
```
x["review"][2]
```

Out[ ]: 'I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer). While some may be disappointed when they realize this is not Match Point 2: Risk Addiction, I thought it was proof that Woody Allen is still fully in control of the style many of us have grown to love.

This was the most I\'d laughed at one of Woody\'s comedies in years (dare I say a decade?). While I\'ve never been impressed with Scarlet Johanson, in this she managed to tone down her "sexy" image and jumped right into a average, but spirited young woman.

This may not be the crown jewel of his career, but it was wittier than "Devil Wears Prada" and more interesting than "Superman" a great comedy to go see with friends.'

In [ ]:

```python
# applying preprocessing to text using nltk and nlp on a copy of X
import nltk
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]: True

In [ ]:

```python
'''
The whole purpose of one hot representation and embedding the corpus is t
which will helps us to make predictions in our neural network.
In case of Machine learning, we can simply use the one hot repr as a data
'''
messages = x.copy()
```

In [ ]:

```python
messages
```

Out[ ]:

	review
0	One of the other reviewers has mentioned that ...
1	A wonderful little production. The...
2	I thought this was a wonderful way to spend ti...
3	Basically there's a family where a little boy ...
4	Petter Mattei's "Love in the Time of Money" is...
...	...
49995	I thought this movie did a down right good job...
49996	Bad plot, bad dialogue, bad acting, idiotic di...
49997	I am a Catholic taught in parochial elementary...
49998	I'm going to have to disagree with the previou...
49999	No one expects the Star Trek movies to be high...

50000 rows × 1 columns

In [ ]:
```
# from nltk.corpus import stopwords
# from nltk.stem.porter import PorterStemmer
# from tqdm import tqdm  # Import tqdm for the progress bar
# import re

# # Assuming 'messages' is your DataFrame containing the 'review' column
# # Replace it with your actual DataFrame and column names if needed

# ps = PorterStemmer()
# corpus = []

# # Use tqdm as a wrapper for your loop to show the progress bar
# for i in tqdm(range(0, len(messages)), desc="Processing reviews"):
#     review = re.sub('[^a-zA-Z]', ' ', messages['review'][i])
#     review = review.lower()
#     review = review.split()

#     review = [ps.stem(word) for word in review if not word in stopwords
#     review = ' '.join(review)
#     corpus.append(review)
```

In [ ]:
```
%store -r corpus
corpus[1]
```

Out[ ]:
```
'wonder littl product br br film techniqu unassum old time bbc fashion g
ive comfort sometim discomfort sens realism entir piec br br actor extre
m well chosen michael sheen got polari voic pat truli see seamless edit
guid refer william diari entri well worth watch terrificli written perfo
rm piec master product one great master comedi life br br realism realli
come home littl thing fantasi guard rather use tradit dream techniqu rem
ain solid disappear play knowledg sens particularli scene concern orton
halliwel set particularli flat halliwel mural decor everi surfac terribl
well done'
```

In [ ]:

In [ ]:
```
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
voc_size = 10000
```

In [ ]:
```
# Applying the one hot repr on corpus
onehot_repr=[one_hot(words,voc_size)for words in corpus]
```

In [ ]:
```
# this is to check the maximum and minimum length of the sentences to fin
len(max(onehot_repr))
```

Out[ ]:
```
128
```

In [ ]:
```
len(min(onehot_repr))
```

Out[ ]:
```
73
```

In [ ]:
```
# applying post padding to make sentences equal length
sent_length=128
embedded_docs=pad_sequences(onehot_repr,padding='post',maxlen=sent_length
```

In [ ]:
```
# padding is applied
embedded_docs[1]
```

Out[ ]:
```
array([3270,  145, 1993, 7803, 7803, 4415, 2665, 3417, 2707,  382, 5111,
        905, 1962, 5763, 9934, 4154, 8314, 1050, 3207, 8694, 7803, 7803,
       5878, 3705, 2979, 3206, 4685, 9944, 5107, 4046, 5059, 4531, 4666,
       9660, 8045,  322, 9406, 7602, 4437, 8705,   91, 2979, 7858, 7243,
       3151, 4754, 4242, 8694, 9625, 1993, 7120, 9346, 9625, 9829, 6327,
       7803, 7803, 1050, 2178,  247, 5611,  145, 6799, 9200, 6059, 4131,
       6138, 2226,  549, 2665, 1345, 6580, 4190, 6643, 2935, 8314, 6885,
       5889, 9271,  277, 2125, 6503, 6885, 6672, 2125, 5605, 7035, 9051,
       9756, 2572, 2979, 4774,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0], dtype=int32)
```

In [ ]:
```
from keras.layers import Dropout
embedding_vector_features = 100
model = Sequential()
model.add(Embedding(voc_size, embedding_vector_features, input_length=sen
model.add(LSTM(units=100,return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(units=100,return_sequences=False))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc
print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 128, 70)           700000

 lstm_3 (LSTM)               (None, 128, 100)          68400

 dropout_2 (Dropout)         (None, 128, 100)          0

 lstm_4 (LSTM)               (None, 100)               80400

 dropout_3 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 848901 (3.24 MB)
Trainable params: 848901 (3.24 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [ ]:

```
num_models = 3
models = []

for i in range(num_models):
    model = Sequential()
    model.add(Embedding(voc_size, embedding_vector_features, input_length
    model.add(LSTM(units=100, return_sequences=True))
    model.add(Dropout(0.3))
    model.add(LSTM(units=100, return_sequences=False))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
    models.append(model)
```

In [ ]:

```
import numpy as np
X_final=np.array(embedded_docs)
y_final=np.array(y)
```

In [ ]:

```
X_final.shape,y_final.shape
```

Out[ ]:

```
((50000, 128), (50000, 1))
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, tes
```

In [ ]:

```
for i, model in enumerate(models):
    print(f"Training Model {i + 1}")
    model.fit(X_train, y_train, epochs=25, batch_size=128, validation_dat
```

```
Training Model 1
Epoch 1/25
313/313 [==============================] - 37s 107ms/step - loss: 0.4591 -
accuracy: 0.7552 - val_loss: 0.3688 - val_accuracy: 0.8620
Epoch 2/25
313/313 [==============================] - 17s 53ms/step - loss: 0.2801 -
accuracy: 0.8909 - val_loss: 0.3077 - val_accuracy: 0.8728
Epoch 3/25
313/313 [==============================] - 11s 35ms/step - loss: 0.2329 -
accuracy: 0.9128 - val_loss: 0.3557 - val_accuracy: 0.8612
Epoch 4/25
313/313 [==============================] - 10s 33ms/step - loss: 0.1953 -
accuracy: 0.9276 - val_loss: 0.3592 - val_accuracy: 0.8613
Epoch 5/25
313/313 [==============================] - 9s 28ms/step - loss: 0.1654 - a
ccuracy: 0.9414 - val_loss: 0.3901 - val_accuracy: 0.8570
Epoch 6/25
313/313 [==============================] - 8s 27ms/step - loss: 0.1452 - a
ccuracy: 0.9492 - val_loss: 0.3794 - val_accuracy: 0.8542
Epoch 7/25
313/313 [==============================] - 8s 26ms/step - loss: 0.1244 - a
ccuracy: 0.9568 - val_loss: 0.4346 - val_accuracy: 0.8536
Epoch 8/25
313/313 [==============================] - 9s 28ms/step - loss: 0.1220 - a
ccuracy: 0.9595 - val_loss: 0.5303 - val_accuracy: 0.8502
Epoch 9/25
313/313 [==============================] - 8s 25ms/step - loss: 0.0988 - a
ccuracy: 0.9685 - val_loss: 0.5815 - val_accuracy: 0.8489
Epoch 10/25
313/313 [==============================] - 8s 27ms/step - loss: 0.0846 - a
ccuracy: 0.9742 - val_loss: 0.5030 - val_accuracy: 0.8357
Epoch 11/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0769 - a
ccuracy: 0.9775 - val_loss: 0.6245 - val_accuracy: 0.8447
Epoch 12/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0718 - a
ccuracy: 0.9794 - val_loss: 0.6492 - val_accuracy: 0.8451
Epoch 13/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0734 - a
ccuracy: 0.9790 - val_loss: 0.5444 - val_accuracy: 0.8355
Epoch 14/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0675 - a
ccuracy: 0.9816 - val_loss: 0.6125 - val_accuracy: 0.8347
Epoch 15/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0568 - a
ccuracy: 0.9852 - val_loss: 0.6297 - val_accuracy: 0.8412
Epoch 16/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0541 - a
ccuracy: 0.9863 - val_loss: 0.6581 - val_accuracy: 0.8405
Epoch 17/25
313/313 [==============================] - 8s 24ms/step - loss: 0.0497 - a
ccuracy: 0.9882 - val_loss: 0.6932 - val_accuracy: 0.8403
Epoch 18/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0572 - a
ccuracy: 0.9854 - val_loss: 0.7114 - val_accuracy: 0.8396
Epoch 19/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0537 - a
ccuracy: 0.9862 - val_loss: 0.6580 - val_accuracy: 0.8355
Epoch 20/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0405 - a
```

```
ccuracy: 0.9904 - val_loss: 0.6694 - val_accuracy: 0.8326
Epoch 21/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0876 - a
ccuracy: 0.9742 - val_loss: 0.5399 - val_accuracy: 0.8254
Epoch 22/25
313/313 [==============================] - 7s 22ms/step - loss: 0.1042 - a
ccuracy: 0.9679 - val_loss: 0.6964 - val_accuracy: 0.8344
Epoch 23/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0541 - a
ccuracy: 0.9865 - val_loss: 0.6643 - val_accuracy: 0.8383
Epoch 24/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0454 - a
ccuracy: 0.9887 - val_loss: 0.6718 - val_accuracy: 0.8333
Epoch 25/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0520 - a
ccuracy: 0.9858 - val_loss: 0.6345 - val_accuracy: 0.8363
Training Model 2
Epoch 1/25
313/313 [==============================] - 36s 103ms/step - loss: 0.4957 -
accuracy: 0.7387 - val_loss: 0.3705 - val_accuracy: 0.8488
Epoch 2/25
313/313 [==============================] - 17s 54ms/step - loss: 0.2971 -
accuracy: 0.8840 - val_loss: 0.3228 - val_accuracy: 0.8717
Epoch 3/25
313/313 [==============================] - 11s 34ms/step - loss: 0.2428 -
accuracy: 0.9089 - val_loss: 0.3083 - val_accuracy: 0.8696
Epoch 4/25
313/313 [==============================] - 9s 30ms/step - loss: 0.2081 - a
ccuracy: 0.9244 - val_loss: 0.3361 - val_accuracy: 0.8604
Epoch 5/25
313/313 [==============================] - 9s 27ms/step - loss: 0.1820 - a
ccuracy: 0.9356 - val_loss: 0.3498 - val_accuracy: 0.8604
Epoch 6/25
313/313 [==============================] - 8s 26ms/step - loss: 0.1492 - a
ccuracy: 0.9482 - val_loss: 0.4164 - val_accuracy: 0.8524
Epoch 7/25
313/313 [==============================] - 9s 28ms/step - loss: 0.1292 - a
ccuracy: 0.9573 - val_loss: 0.3986 - val_accuracy: 0.8547
Epoch 8/25
313/313 [==============================] - 9s 29ms/step - loss: 0.1083 - a
ccuracy: 0.9659 - val_loss: 0.4615 - val_accuracy: 0.8583
Epoch 9/25
313/313 [==============================] - 8s 25ms/step - loss: 0.1009 - a
ccuracy: 0.9682 - val_loss: 0.4625 - val_accuracy: 0.8396
Epoch 10/25
313/313 [==============================] - 7s 24ms/step - loss: 0.0921 - a
ccuracy: 0.9725 - val_loss: 0.4641 - val_accuracy: 0.8510
Epoch 11/25
313/313 [==============================] - 7s 24ms/step - loss: 0.0833 - a
ccuracy: 0.9762 - val_loss: 0.5204 - val_accuracy: 0.8446
Epoch 12/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0721 - a
ccuracy: 0.9798 - val_loss: 0.4919 - val_accuracy: 0.8400
Epoch 13/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0644 - a
ccuracy: 0.9833 - val_loss: 0.6471 - val_accuracy: 0.8494
Epoch 14/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0649 - a
ccuracy: 0.9826 - val_loss: 0.6161 - val_accuracy: 0.8458
Epoch 15/25
```

```
313/313 [==============================] - 7s 21ms/step - loss: 0.6525 - a
ccuracy: 0.5840 - val_loss: 0.6057 - val_accuracy: 0.7020
Epoch 16/25
313/313 [==============================] - 7s 22ms/step - loss: 0.6501 - a
ccuracy: 0.6228 - val_loss: 0.6905 - val_accuracy: 0.6548
Epoch 17/25
313/313 [==============================] - 7s 22ms/step - loss: 0.6588 - a
ccuracy: 0.5986 - val_loss: 0.5983 - val_accuracy: 0.7349
Epoch 18/25
313/313 [==============================] - 7s 23ms/step - loss: 0.4376 - a
ccuracy: 0.8187 - val_loss: 0.4469 - val_accuracy: 0.8213
Epoch 19/25
313/313 [==============================] - 7s 23ms/step - loss: 0.3579 - a
ccuracy: 0.8583 - val_loss: 0.3852 - val_accuracy: 0.8408
Epoch 20/25
313/313 [==============================] - 7s 22ms/step - loss: 0.2458 - a
ccuracy: 0.9041 - val_loss: 0.3958 - val_accuracy: 0.8561
Epoch 21/25
313/313 [==============================] - 7s 22ms/step - loss: 0.1834 - a
ccuracy: 0.9333 - val_loss: 0.4289 - val_accuracy: 0.8500
Epoch 22/25
313/313 [==============================] - 7s 23ms/step - loss: 0.1352 - a
ccuracy: 0.9525 - val_loss: 0.4910 - val_accuracy: 0.8503
Epoch 23/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0955 - a
ccuracy: 0.9694 - val_loss: 0.5432 - val_accuracy: 0.8518
Epoch 24/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0700 - a
ccuracy: 0.9797 - val_loss: 0.5787 - val_accuracy: 0.8478
Epoch 25/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0514 - a
ccuracy: 0.9870 - val_loss: 0.6150 - val_accuracy: 0.8463
Training Model 3
Epoch 1/25
313/313 [==============================] - 36s 104ms/step - loss: 0.4587 -
accuracy: 0.7646 - val_loss: 0.3088 - val_accuracy: 0.8669
Epoch 2/25
313/313 [==============================] - 17s 53ms/step - loss: 0.2828 -
accuracy: 0.8899 - val_loss: 0.3308 - val_accuracy: 0.8641
Epoch 3/25
313/313 [==============================] - 10s 33ms/step - loss: 0.2359 -
accuracy: 0.9105 - val_loss: 0.3298 - val_accuracy: 0.8674
Epoch 4/25
313/313 [==============================] - 10s 32ms/step - loss: 0.1990 -
accuracy: 0.9269 - val_loss: 0.3421 - val_accuracy: 0.8613
Epoch 5/25
313/313 [==============================] - 9s 27ms/step - loss: 0.1709 - a
ccuracy: 0.9388 - val_loss: 0.4052 - val_accuracy: 0.8569
Epoch 6/25
313/313 [==============================] - 8s 27ms/step - loss: 0.1433 - a
ccuracy: 0.9504 - val_loss: 0.4356 - val_accuracy: 0.8527
Epoch 7/25
313/313 [==============================] - 7s 24ms/step - loss: 0.1309 - a
ccuracy: 0.9554 - val_loss: 0.4473 - val_accuracy: 0.8444
Epoch 8/25
313/313 [==============================] - 8s 25ms/step - loss: 0.1228 - a
ccuracy: 0.9596 - val_loss: 0.4945 - val_accuracy: 0.8515
Epoch 9/25
313/313 [==============================] - 8s 26ms/step - loss: 0.1015 - a
ccuracy: 0.9674 - val_loss: 0.4622 - val_accuracy: 0.8499
```

```
Epoch 10/25
313/313 [==============================] - 8s 27ms/step - loss: 0.0922 - a
ccuracy: 0.9712 - val_loss: 0.4772 - val_accuracy: 0.8507
Epoch 11/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0772 - a
ccuracy: 0.9768 - val_loss: 0.6146 - val_accuracy: 0.8401
Epoch 12/25
313/313 [==============================] - 7s 24ms/step - loss: 0.0775 - a
ccuracy: 0.9774 - val_loss: 0.5860 - val_accuracy: 0.8420
Epoch 13/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0722 - a
ccuracy: 0.9803 - val_loss: 0.6007 - val_accuracy: 0.8445
Epoch 14/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0755 - a
ccuracy: 0.9777 - val_loss: 0.6405 - val_accuracy: 0.8448
Epoch 15/25
313/313 [==============================] - 8s 24ms/step - loss: 0.0644 - a
ccuracy: 0.9823 - val_loss: 0.6107 - val_accuracy: 0.8439
Epoch 16/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0574 - a
ccuracy: 0.9851 - val_loss: 0.6661 - val_accuracy: 0.8434
Epoch 17/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0508 - a
ccuracy: 0.9872 - val_loss: 0.6428 - val_accuracy: 0.8385
Epoch 18/25
313/313 [==============================] - 7s 21ms/step - loss: 0.0478 - a
ccuracy: 0.9880 - val_loss: 0.7205 - val_accuracy: 0.8428
Epoch 19/25
313/313 [==============================] - 7s 23ms/step - loss: 0.0997 - a
ccuracy: 0.9705 - val_loss: 0.5165 - val_accuracy: 0.8276
Epoch 20/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0753 - a
ccuracy: 0.9778 - val_loss: 0.6156 - val_accuracy: 0.8324
Epoch 21/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0587 - a
ccuracy: 0.9841 - val_loss: 0.6409 - val_accuracy: 0.8409
Epoch 22/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0399 - a
ccuracy: 0.9903 - val_loss: 0.6775 - val_accuracy: 0.8432
Epoch 23/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0367 - a
ccuracy: 0.9921 - val_loss: 0.7137 - val_accuracy: 0.8346
Epoch 24/25
313/313 [==============================] - 7s 21ms/step - loss: 0.0420 - a
ccuracy: 0.9897 - val_loss: 0.7537 - val_accuracy: 0.8419
Epoch 25/25
313/313 [==============================] - 7s 22ms/step - loss: 0.0414 - a
ccuracy: 0.9894 - val_loss: 0.7179 - val_accuracy: 0.8397
```

In [ ]:

```
predictions = np.zeros_like(y_test, dtype=float)
for model in models:
    predictions += model.predict(X_test)
```

```
313/313 [==============================] - 3s 7ms/step
313/313 [==============================] - 3s 7ms/step
313/313 [==============================] - 3s 7ms/step
```

In [ ]:

```
# Average the predictions from all models
ensemble_prediction = predictions / num_models
```

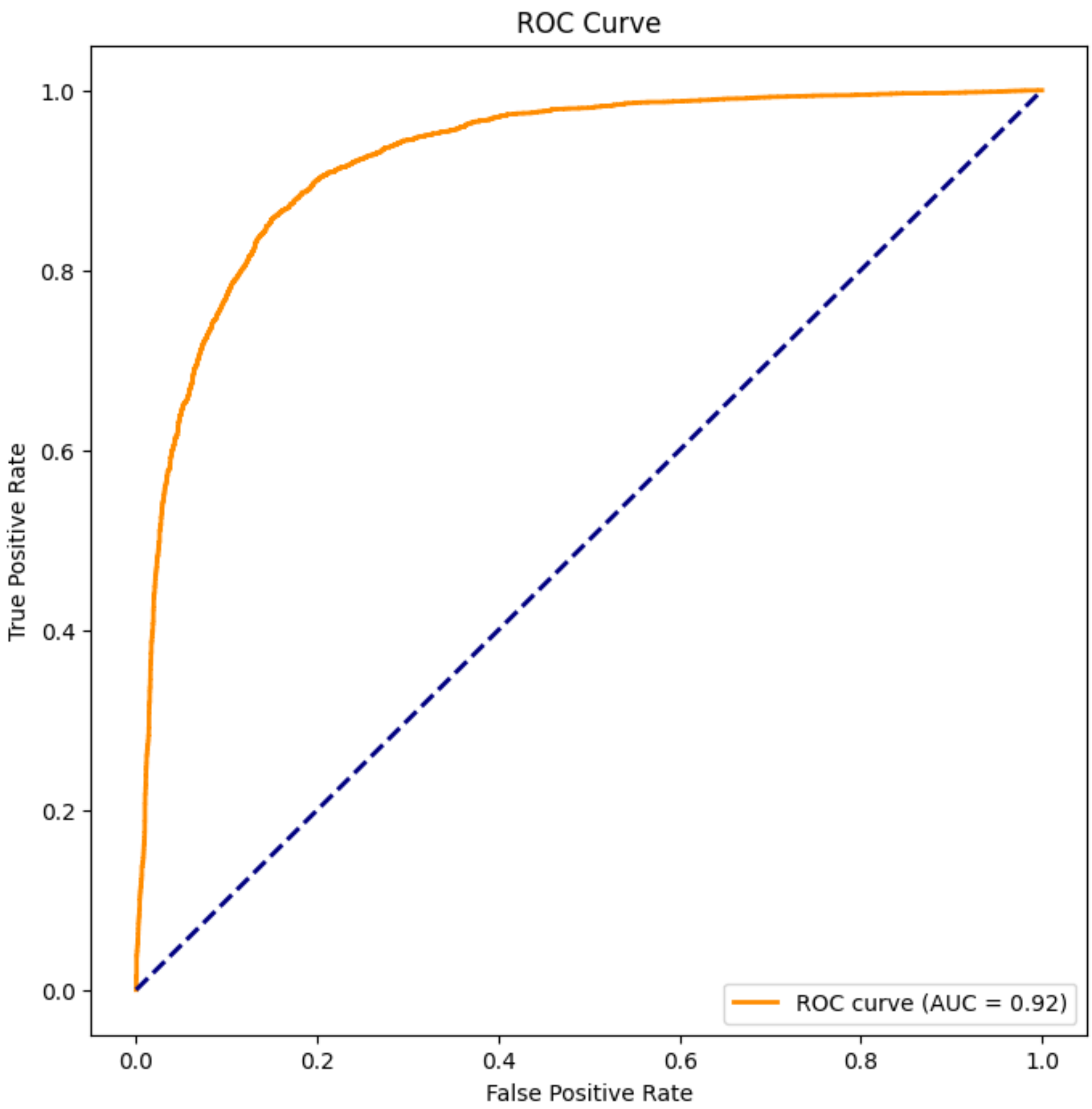
In [ ]:
```
ensemble_prediction_binary = (ensemble_prediction > 0.5).astype(int)
```

In [ ]:
```
# Evaluate the ensemble on the validation set
accuracy = np.mean(ensemble_prediction_binary == y_test)
print(f"Ensemble Accuracy: {accuracy}")
```

```
Ensemble Accuracy: 0.8511
```

In [ ]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

# Assuming you have X_test, y_test, and models from the previous code

# Make predictions on the test set with each model
predictions = np.zeros_like(y_test, dtype=float)
for model in models:
    predictions += model.predict(X_test)

# Average the predictions from all models
ensemble_prediction = predictions / num_models

# Calculate the ROC curve and AUC for the ensemble
fpr, tpr, thresholds = roc_curve(y_test, ensemble_prediction)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = {:.2
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.savefig("auc2.png")
plt.show()
```

```
313/313 [==============================] - 2s 7ms/step
313/313 [==============================] - 2s 7ms/step
313/313 [==============================] - 2s 7ms/step
```


ROC Curve
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.92)
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]:
```python
from sklearn.metrics import confusion_matrix, classification_report


# Make predictions on the test set with each model
predictions = np.zeros_like(y_test, dtype=float)
for model in models:
    predictions += model.predict(X_test)

# Average the predictions from all models
ensemble_prediction = (predictions / num_models > 0.5).astype(int)

# Create confusion matrix
conf_matrix = confusion_matrix(y_test, ensemble_prediction)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

# Create classification report
class_report = classification_report(y_test, ensemble_prediction)

# Print classification report
print("\nClassification Report:")
print(class_report)
```

```
313/313 [==============================] - 2s 7ms/step
313/313 [==============================] - 2s 7ms/step
313/313 [==============================] - 2s 7ms/step
Confusion Matrix:
[[4125  836]
 [ 653 4386]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.83      0.85      4961
           1       0.84      0.87      0.85      5039

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```

In [ ]: