In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
from keras.datasets import imdb
```

In [3]:

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
data = np.concatenate((X_train,X_test), axis=0)
label = np.concatenate((y_train, y_test), axis=0)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/imdb.npz (https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/imdb.npz)
17464789/17464789 [==============================] - 1s 0us/step
```

In [4]:

```
print("Review is ", X_train[5])
print("Review is ", X_test[5])
```

```
Review is  [1, 778, 128, 74, 12, 630, 163, 15, 4, 1766, 7982, 1051, 2,
32, 85, 156, 45, 40, 148, 139, 121, 664, 665, 10, 10, 1361, 173, 4, 74
9, 2, 16, 3804, 8, 4, 226, 65, 12, 43, 127, 24, 2, 10, 10]
Review is  [1, 146, 427, 5718, 14, 20, 218, 112, 2962, 32, 37, 119, 1
4, 20, 144, 9493, 910, 5, 8817, 4, 4659, 18, 12, 3403, 853, 28, 8, 222
5, 12, 95, 474, 818, 4651, 18, 1462, 13, 124, 285, 5, 1462, 11, 14, 2
0, 122, 6, 52, 292, 5, 13, 774, 2626, 46, 138, 910, 1481, 276, 14, 20,
23, 288, 42, 23, 1856, 11, 2364, 5687, 33, 222, 13, 774, 110, 101, 465
1, 14, 9, 6, 3799, 52, 20, 5, 144, 30, 110, 34, 32, 4, 362, 11, 4, 16
2, 2248, 92, 79, 8, 67, 12, 5, 13, 104, 36, 144, 12, 144, 33, 222, 30,
276, 145, 23, 4, 1308, 14, 20, 152, 1833, 6, 706, 2, 12, 1015, 4, 147,
155, 146, 98, 150, 14, 20, 80, 30, 23, 288]
```

In [5]:

```
vocab=imdb.get_word_index()
print(vocab)
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json (https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json)

```
1641221/1641221 [==============================] - 1s 0us/step
```

{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders': 16115, 'hanging': 2345, 'woody': 2289, 'trawling': 52008, "hold's": 52009, 'comically': 11307, 'localized': 40830, 'disobeying': 30568, "'royale": 52010, "harpo's": 40831, 'canet': 52011, 'aileen': 19313, 'acurately': 52012, "diplomat's": 52013, 'rickman': 25242, 'arranged': 6746, 'rumbustious': 52014, 'familiarness': 52015, "spider'": 52016, 'hahahah': 68804, "wood'": 52017, 'transvestism': 40833, "hangin'": 34702, 'bringing': 2338, 'seamier': 40834, 'wooded': 34703, 'bravora': 52018, 'grueling': 16817, 'wooden': 1636, 'wednesday': 16818, "'prix": 52019, 'altagracia': 34704, 'circuitry': 52020, 'crotch': 11585, 'busybody': 57766, "tart'n'tangy": 52021, 'burgade': 14129, 'thrace': 52023, "tom's": 11038, 'snuggles': 52025, 'francesco': 29114, 'complainers': 52027, 'templarios': 52125, '272': 40835, '273': 52028, 'zaniacs': 52130, '275': 34706, 'consenting': 27631, 'snuggled': 40836, 'inani

In [6]:

```
data
```

Out[6]:

```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 6
6, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35,
480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 111
1, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4,
1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247,
4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 10
6, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619,
5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52,
5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766,
5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 10
4, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18,
4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 2
24, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 447
2, 113, 103, 32, 15, 16, 5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012,
134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5,
207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 11
4, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322,
4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455,
9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215,
2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15,
123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 11
57, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 82
55, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464,
1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 169
0, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145,
95]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61,
369, 13, 71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43,
1829, 296, 4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89,
78, 12, 66, 16, 4, 360, 7, 4, 58, 316, 334, 11, 4, 1716, 43, 645, 662,
8, 257, 85, 1200, 42, 1228, 2578, 83, 68, 3912, 15, 36, 165, 1539, 27
8, 36, 69, 2, 780, 8, 106, 14, 6905, 1338, 18, 6, 22, 12, 215, 28, 61
0, 40, 6, 87, 326, 23, 2300, 21, 23, 22, 12, 272, 40, 57, 31, 11, 4, 2
2, 47, 6, 2307, 51, 9, 170, 23, 595, 116, 595, 1352, 13, 191, 79, 638,
89, 2, 14, 9, 8, 106, 607, 624, 35, 534, 6, 227, 7, 129, 113]),
       ...,
       list([1, 13, 1408, 15, 8, 135, 14, 9, 35, 32, 46, 394, 20, 62,
30, 5093, 21, 45, 184, 78, 4, 1492, 910, 769, 2290, 2515, 395, 4257,
5, 1454, 11, 119, 2, 89, 1036, 4, 116, 218, 78, 21, 407, 100, 30, 128,
262, 15, 7, 185, 2280, 284, 1842, 2, 37, 315, 4, 226, 20, 272, 2942, 4
0, 29, 152, 60, 181, 8, 30, 50, 553, 362, 80, 119, 12, 21, 846, 551
8]),
       list([1, 11, 119, 241, 9, 4, 840, 20, 12, 468, 15, 94, 3684, 56
2, 791, 39, 4, 86, 107, 8, 97, 14, 31, 33, 4, 2960, 7, 743, 46, 1028,
9, 3531, 5, 4, 768, 47, 8, 79, 90, 145, 164, 162, 50, 6, 501, 119, 7,
9, 4, 78, 232, 15, 16, 224, 11, 4, 333, 20, 4, 985, 200, 5, 2, 5, 9, 1
861, 8, 79, 357, 4, 20, 47, 220, 57, 206, 139, 11, 12, 5, 55, 117, 21
2, 13, 1276, 92, 124, 51, 45, 1188, 71, 536, 13, 520, 14, 20, 6, 2302,
7, 470]),
       list([1, 6, 52, 7465, 430, 22, 9, 220, 2594, 8, 28, 2, 519, 322
7, 6, 769, 15, 47, 6, 3482, 4067, 8, 114, 5, 33, 222, 31, 55, 184, 70
4, 5586, 2, 19, 346, 3153, 5, 6, 364, 350, 4, 184, 5586, 9, 133, 1810,
11, 5417, 2, 21, 4, 7298, 2, 570, 50, 2005, 2643, 9, 6, 1249, 17, 6,
2, 2, 21, 17, 6, 1211, 232, 1138, 2249, 29, 266, 56, 96, 346, 194, 30
```

```
8, 9, 194, 21, 29, 218, 1078, 19, 4, 78, 173, 7, 27, 2, 5698, 3406, 71
8, 2, 9, 6, 6907, 17, 210, 5, 3281, 5677, 47, 77, 395, 14, 172, 173, 1
8, 2740, 2931, 4517, 82, 127, 27, 173, 11, 6, 392, 217, 21, 50, 9, 57,
65, 12, 2, 53, 40, 35, 390, 7, 11, 4, 3567, 7, 4, 314, 74, 6, 792, 22,
2, 19, 714, 727, 5205, 382, 4, 91, 6533, 439, 19, 14, 20, 9, 1441, 580
5, 1118, 4, 756, 25, 124, 4, 31, 12, 16, 93, 804, 34, 2005, 2643])],
      dtype=object)
```

In [7]:

```
label
```

Out[7]:

```
array([1, 0, 0, ..., 0, 0, 0])
```

In [11]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(25000,)
(25000,)
(25000,)
(25000,)
```

In [12]:

```
def vectorize(sequences, dimension = 10000):
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1
  return results
```

In [14]:

```
test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]
```

In [18]:

```
print(test_x, test_y, train_x, train_y)
```

```
[list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3
941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 4
80, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 11
11, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22,
4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13,
1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 31
6, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16,
38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215,
28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4,
2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2,
1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 19
4, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 1
8, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 1
6, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32])
 list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 13
4, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5,
207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 1
14, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 132
2, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4,
```

In [19]:

```
print("Categories:", np.unique(label))
print("Number of unique words:", len(np.unique(np.hstack(data))))
```

```
Categories: [0 1]
Number of unique words: 9998
```

In [20]:

```
length = [len(i) for i in data]
print("Average Review length: ", np.mean(length))
print("Standard Deviation: ", round(np.std(length)))
```

```
Average Review length:  234.75892
Standard Deviation:  173
```

In [22]:

```
print("Label:", label[0])
```

```
Label: 1
```

In [23]:

```
print(data[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4,
173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284,
5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 54
6, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 461
3, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 1
7, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4,
2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 12
4, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 4
07, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 3
6, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,
381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22,
21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25,
104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103,
32, 15, 16, 5345, 19, 178, 32]
```

In [26]:

```
index= imdb.get_word_index()
reverse_index = dict([(value, key) for (key,value) in index.items()])
decoded = " ".join([reverse_index.get(i - 3,"#") for i in data[0]])
print(decoded)
```

```
# this film was just brilliant casting location scenery story directio
n everyone's really suited the part they played and you could just ima
gine being there robert # is an amazing actor and now the same being d
irector # father came from the same scottish island as myself so i lov
ed the fact there was a real connection with this film the witty remar
ks throughout the film were great it was just brilliant so much that i
bought the film as soon as it was released for # and would recommend i
t to everyone to watch and the fly fishing was amazing really cried at
the end it was so sad and you know what they say if you cry at a film
it must have been good and this definitely was also # to the two littl
e boy's that played the # of norman and paul they were just brilliant
children are often left out of the # list i think because the stars th
at play them all grown up are such a big profile for the whole film bu
t these children are amazing and should be praised for what they have
done don't you think the whole story was so lovely because it was true
and was someone's life after all that was shared with us all
```

In [28]:

```
# print(index, reverse_index, decoded)
```

In [29]:

```
data = vectorize(data)
label= np.array(label).astype("float32")
```

In [30]:

```
print(data)
print(label)
```

```
[[0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]
 ...
 [0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]]
[1. 0. 0. ... 0. 0. 0.]
```

In [31]:

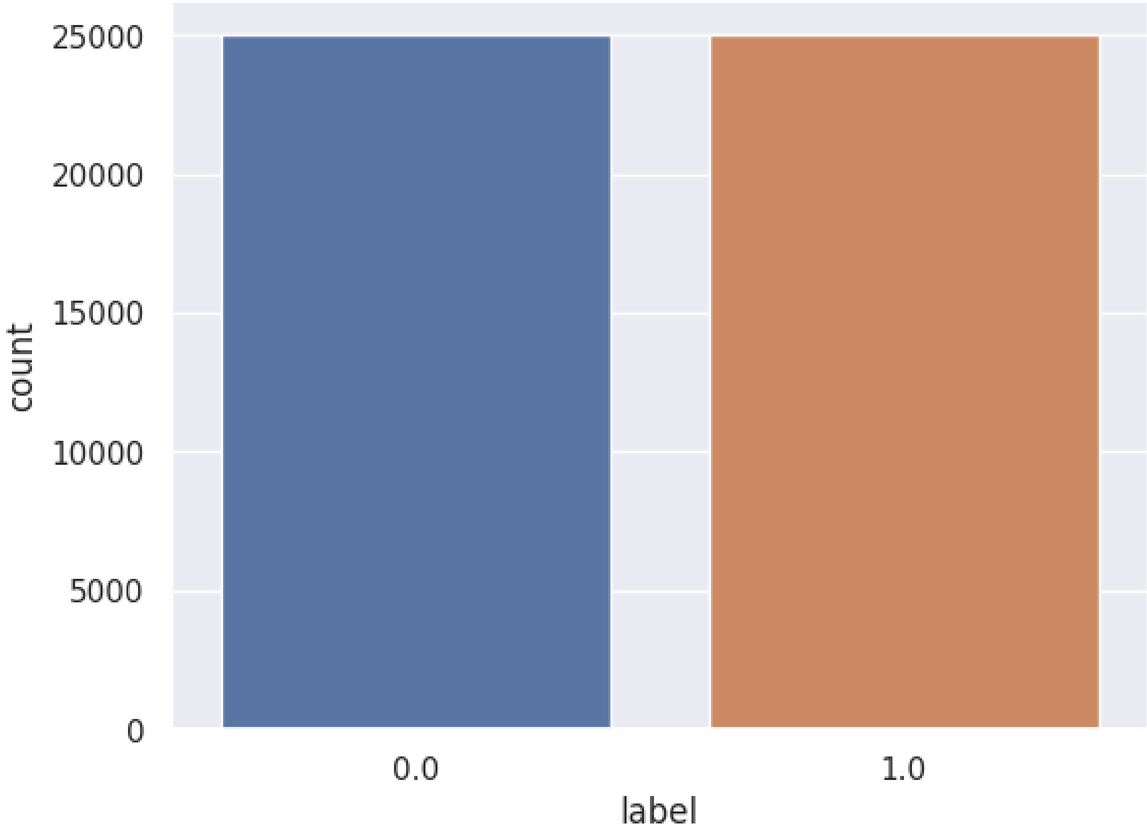
```
import seaborn as sns
sns.set(color_codes=True)
import matplotlib.pyplot as plt
%matplotlib inline
```

In [32]:

```
labelDF=pd.DataFrame({'label':label})
sns.countplot(x='label', data=labelDF)
```

Out[32]:

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



In [33]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.20, ra
```

In [34]:

```
X_train.shape
X_test.shape
```

Out[34]:

```
(10000, 10000)
```

In [35]:

```
from keras.utils import to_categorical
from keras import models
from keras import layers
```

In [36]:

```
model = models.Sequential()
model.add(layers.Dense(50, activation = 'relu', input_shape=(10000, )))
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = 'relu'))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 50)                500050    
                                                                 
 dropout (Dropout)           (None, 50)                0         
                                                                 
 dense_1 (Dense)             (None, 50)                2550      
                                                                 
 dropout_1 (Dropout)         (None, 50)                0         
                                                                 
 dense_2 (Dense)             (None, 50)                2550      
                                                                 
 dense_3 (Dense)             (None, 1)                 51        
                                                                 
=================================================================
Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```
import tensorflow as tf
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

model.compile(optimizer='adam',
              loss = 'binary_crossentropy',
              metrics = ["accuracy"])
```

In [38]:

```
results = model.fit(
    X_train, y_train,
    epochs = 2,
    batch_size = 500,
    validation_data= (X_test, y_test),
    callbacks=[callback]
)
```

```
Epoch 1/2
80/80 [==============================] - 8s 78ms/step - loss: 0.4090 -
accuracy: 0.8188 - val_loss: 0.2551 - val_accuracy: 0.8989
Epoch 2/2
80/80 [==============================] - 8s 96ms/step - loss: 0.2186 -
accuracy: 0.9165 - val_loss: 0.2552 - val_accuracy: 0.8966
```

In [39]:

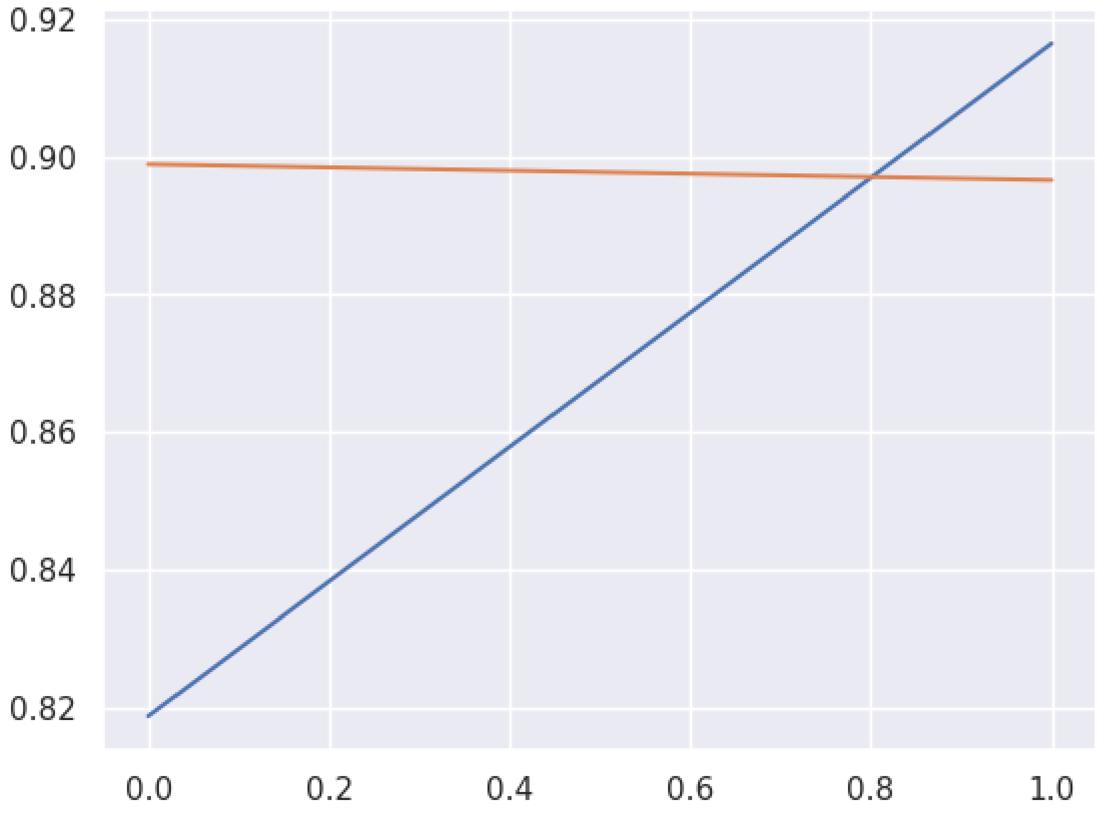
```
print(np.mean(results.history["val_accuracy"]))

print(results.history.keys())
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
```

```
0.8977499902248383
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Out[39]:

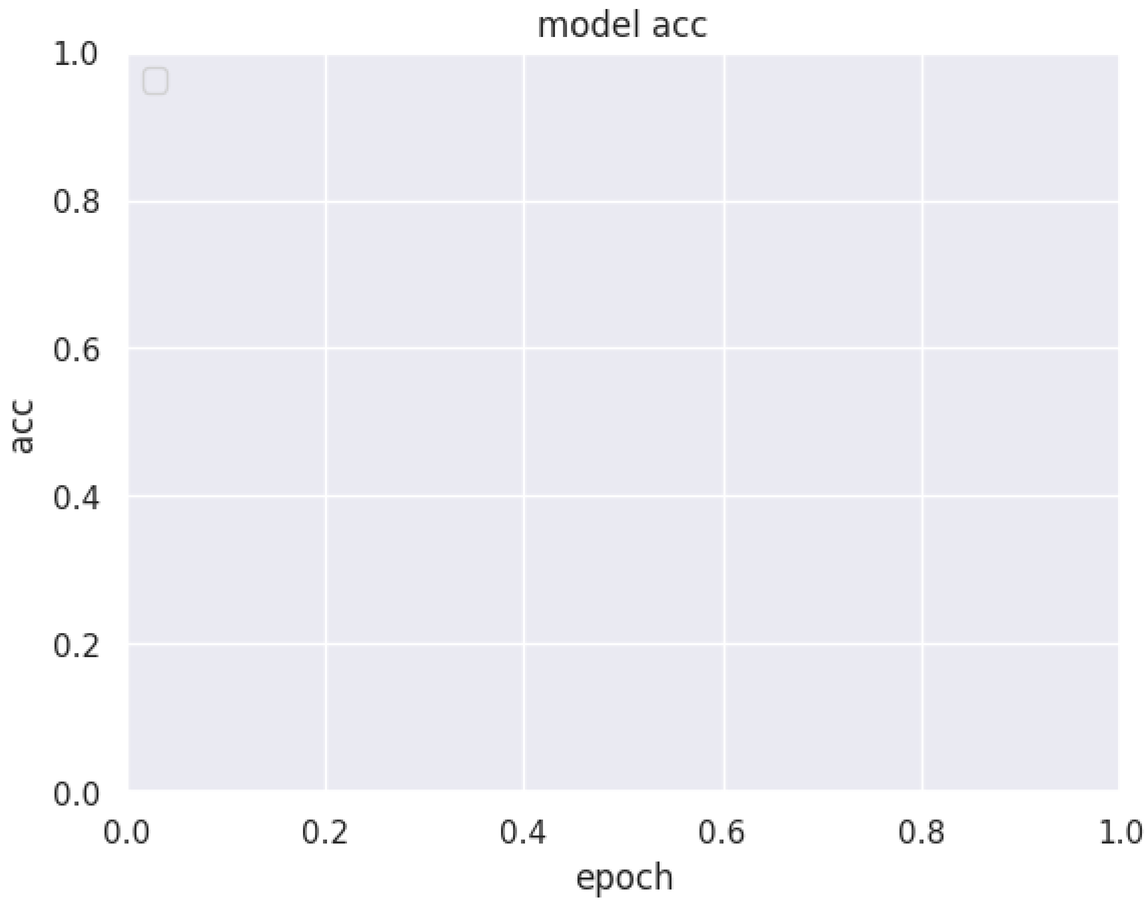
```
[<matplotlib.lines.Line2D at 0x7f47d1defb20>]
```

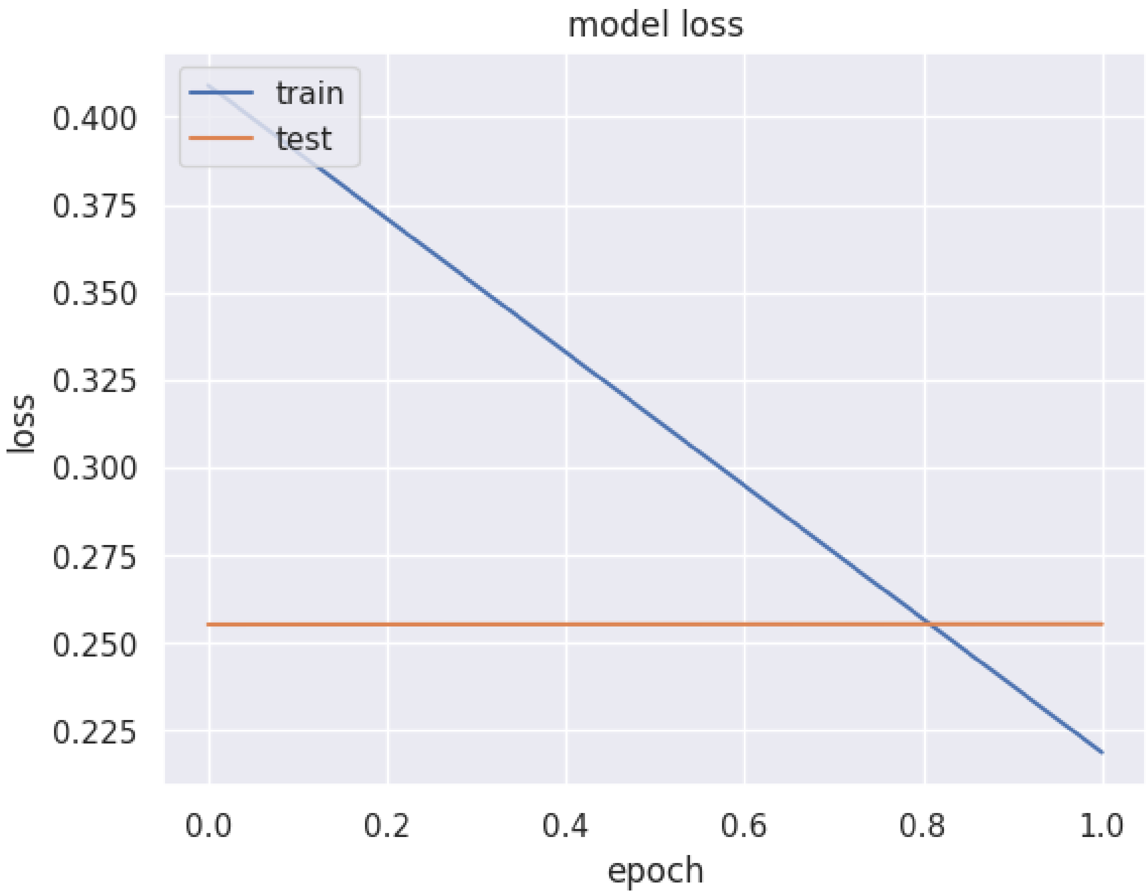


In [43]:

```
plt.title('model acc')
plt.ylabel('acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




model loss
train
test
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
loss
0.0
0.2
0.4
0.6
0.8
1.0
epoch

In [43]:

```

```