Importing necessary packages

In [ ]:
```
import numpy as np
import pandas as pd
```

STEP -1 import and analyze the data set

In [ ]:
```
#loading imdb data with most frequent 10000 words

from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 3s 0us/step
```

Let's check dimentions of dataset

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(25000,)
```

In [ ]:
```
X_test.shape
```

Out[ ]:
```
(25000,)
```

Function to perform relevant sequence adding on the data

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

In [ ]:
```
#consolidating data for EDA
data = np.concatenate((X_train, X_test), axis=0)
label = np.concatenate((y_train, y_test), axis=0)
```

In [ ]:
```
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 [ ]:
```
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
```

Let's look at a single training example:

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

```
Label: 1
```

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

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 10
0, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 1
72, 4536, 1111, 17, 546, 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, 17, 515, 17, 12, 16, 62
6, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 1
2, 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, 2
6, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 1
94, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 2
5, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 1
9, 178, 32]
```

Let's decode the first review

In [ ]:
```
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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_i
ndex.json
1641221/1641221 [==============================] - 0s 0us/step
# this film was just brilliant casting location scenery story direction everyone's really sui
ted the part they played and you could just imagine being there robert # is an amazing actor
and now the same being director # father came from the same scottish island as myself so i lo
ved the fact there was a real connection with this film the witty remarks throughout the film
were great it was just brilliant so much that i bought the film as soon as it was released fo
r # and would recommend it 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 go
od and this definitely was also # to the two little boy's that played the # of norman and pau
l they were just brilliant children are often left out of the # list i think because the star
s that play them all grown up are such a big profile for the whole film but these children ar
e amazing and should be praised for what they have done don't you think the whole story was s
o lovely because it was true and was someone's life after all that was shared with us all
```

In [ ]:
```
#Adding sequence to data
data = vectorize(data)
label = np.array(label).astype("float32")
```

In [ ]:
```
label
```

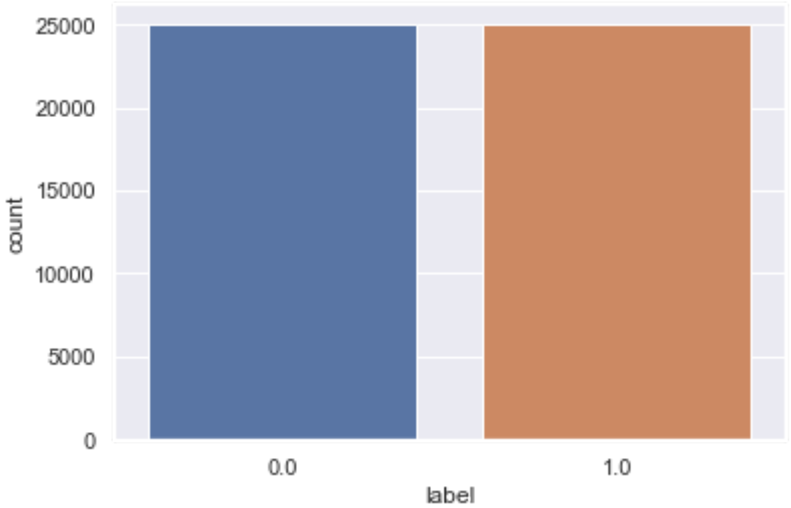
Out[ ]:
```
array([1., 0., 0., ..., 0., 0., 0.], dtype=float32)
```

Let's check distribution of data

In [ ]:
```
#To plot for EDA
import seaborn as sns
sns.set(color_codes=True)
import matplotlib.pyplot as plt
%matplotlib inline
```

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

Out[ ]:
```
<AxesSubplot:xlabel='label', ylabel='count'>
```



For above analysis it is clear that data has equel distribution of sentiments.This will help us building a good model.

Creating train and test data set

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,label, test_size=0.30, random_state=
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(35000, 10000)
```

In [ ]:
```
X_test.shape
```

Out[ ]:
```
(15000, 10000)
```

Let's create sequential model

In [ ]:
```
from keras.utils import to_categorical
from keras import models
from keras import layers
```

In [ ]:
```
model = models.Sequential()
# Input - Layer
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))
# Hidden - Layers
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"))
# Output- Layer
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         
                                                                 
```



```
_________________________________________________________________
 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 [ ]:

```python
#For early stopping
import tensorflow as tf
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```

In [ ]:

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

In [ ]:

```python
results = model.fit(
 X_train, y_train,
 epochs= 100,
 batch_size = 40,
 validation_data = (X_test, y_test),
 callbacks=[callback]
)
```

```
Epoch 1/100
875/875 [==============================] - 15s 16ms/step - loss: 0.3251 - accuracy: 0.8618 -
val_loss: 0.2637 - val_accuracy: 0.8905
Epoch 2/100
875/875 [==============================] - 9s 10ms/step - loss: 0.1984 - accuracy: 0.9229 - v
al_loss: 0.2742 - val_accuracy: 0.8889
Epoch 3/100
875/875 [==============================] - 9s 10ms/step - loss: 0.1378 - accuracy: 0.9483 - v
al_loss: 0.3456 - val_accuracy: 0.8805
Epoch 4/100
875/875 [==============================] - 8s 9ms/step - loss: 0.1015 - accuracy: 0.9619 - va
l_loss: 0.3908 - val_accuracy: 0.8819
Epoch 5/100
875/875 [==============================] - 9s 10ms/step - loss: 0.0765 - accuracy: 0.9703 - v
al_loss: 0.4354 - val_accuracy: 0.8813
Epoch 6/100
875/875 [==============================] - 9s 10ms/step - loss: 0.0640 - accuracy: 0.9765 - v
al_loss: 0.4764 - val_accuracy: 0.8799
Epoch 7/100
875/875 [==============================] - 8s 10ms/step - loss: 0.0590 - accuracy: 0.9766 - v
al_loss: 0.4969 - val_accuracy: 0.8815
Epoch 8/100
875/875 [==============================] - 8s 10ms/step - loss: 0.0493 - accuracy: 0.9811 - v
al_loss: 0.5938 - val_accuracy: 0.8842
Epoch 9/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0435 - accuracy: 0.9838 - va
l_loss: 0.6029 - val_accuracy: 0.8815
Epoch 10/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0428 - accuracy: 0.9829 - va
l_loss: 0.5171 - val_accuracy: 0.8809
Epoch 11/100
875/875 [==============================] - 10s 11ms/step - loss: 0.0394 - accuracy: 0.9847 -
val_loss: 0.5615 - val_accuracy: 0.8804
Epoch 12/100
875/875 [==============================] - 8s 10ms/step - loss: 0.0385 - accuracy: 0.9845 - v
al_loss: 0.5879 - val_accuracy: 0.8831
Epoch 13/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0323 - accuracy: 0.9869 - va
l_loss: 0.5993 - val_accuracy: 0.8853
Epoch 14/100
875/875 [==============================] - 7s 8ms/step - loss: 0.0340 - accuracy: 0.9860 - va
l_loss: 0.6658 - val_accuracy: 0.8848
Epoch 15/100
875/875 [==============================] - 7s 8ms/step - loss: 0.0338 - accuracy: 0.9866 - va
l_loss: 0.6686 - val_accuracy: 0.8831
Epoch 16/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0286 - accuracy: 0.9886 - va
l_loss: 0.7215 - val_accuracy: 0.8825
Epoch 17/100
875/875 [==============================] - 8s 10ms/step - loss: 0.0277 - accuracy: 0.9881 - v
al_loss: 0.7170 - val_accuracy: 0.8828
Epoch 18/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0294 - accuracy: 0.9888 - va
l_loss: 0.6234 - val_accuracy: 0.8778
Epoch 19/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0266 - accuracy: 0.9894 - va
l_loss: 0.5884 - val_accuracy: 0.8783
Epoch 20/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0253 - accuracy: 0.9888 - va
l_loss: 0.6780 - val_accuracy: 0.8803
Epoch 21/100
875/875 [==============================] - 7s 9ms/step - loss: 0.0248 - accuracy: 0.9896 - va
l_loss: 0.7056 - val_accuracy: 0.8833
Epoch 22/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0269 - accuracy: 0.9887 - va
l_loss: 0.7517 - val_accuracy: 0.8807
Epoch 23/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0256 - accuracy: 0.9899 - va
l_loss: 0.7330 - val_accuracy: 0.8814
```

```
Epoch 24/100
875/875 [==============================] - 7s 8ms/step - loss: 0.0231 - accuracy: 0.9909 - va
l_loss: 0.7312 - val_accuracy: 0.8810
Epoch 25/100
875/875 [==============================] - 8s 10ms/step - loss: 0.0244 - accuracy: 0.9894 - v
al_loss: 0.7113 - val_accuracy: 0.8847
Epoch 26/100
875/875 [==============================] - 9s 10ms/step - loss: 0.0228 - accuracy: 0.9904 - v
al_loss: 0.7679 - val_accuracy: 0.8820
Epoch 27/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0229 - accuracy: 0.9904 - va
l_loss: 0.7327 - val_accuracy: 0.8819
Epoch 28/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0204 - accuracy: 0.9915 - va
l_loss: 0.8254 - val_accuracy: 0.8843
Epoch 29/100
875/875 [==============================] - 7s 8ms/step - loss: 0.0189 - accuracy: 0.9913 - va
l_loss: 0.8224 - val_accuracy: 0.8828
Epoch 30/100
875/875 [==============================] - 7s 8ms/step - loss: 0.0198 - accuracy: 0.9913 - va
l_loss: 0.7843 - val_accuracy: 0.8841
Epoch 31/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0201 - accuracy: 0.9911 - va
l_loss: 0.8507 - val_accuracy: 0.8835
Epoch 32/100
875/875 [==============================] - 8s 9ms/step - loss: 0.0207 - accuracy: 0.9917 - va
l_loss: 0.8062 - val_accuracy: 0.8795
```

Let's check mean accuracy of our model

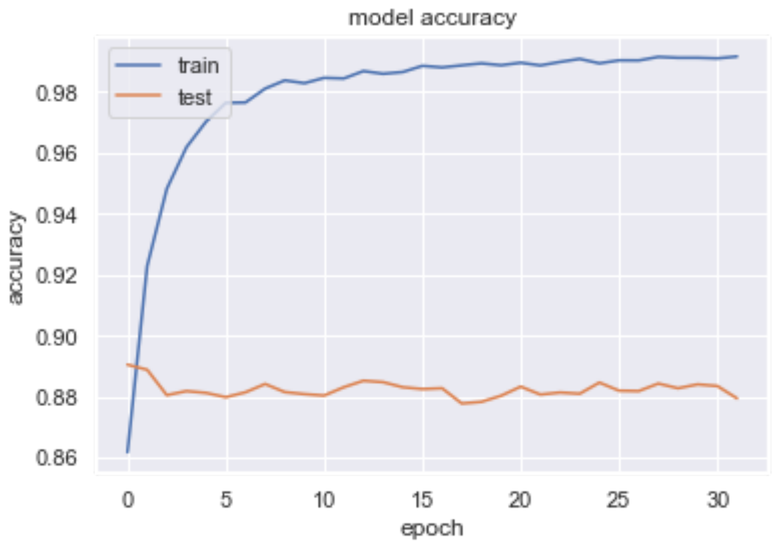
In [ ]:
```
print(np.mean(results.history["val_accuracy"]))
```

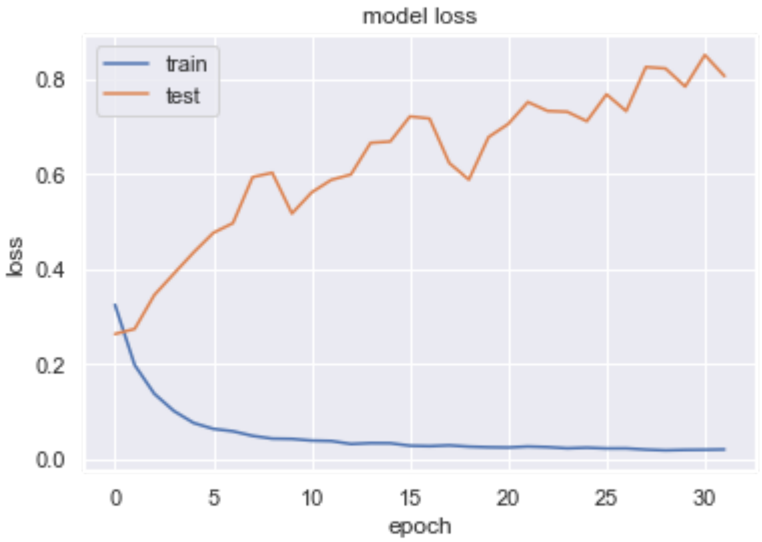
```
0.8824604135006666
```

In [ ]:
```
#Let's plot training history of our model

# list all data in history
print(results.history.keys())
# summarize history for accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
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()
```

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





In [ ]:
```
model.predict(X_test)
```

```
469/469 [==============================] - 1s 2ms/step
```

Out[ ]:
```
array([[1.3325606e-02],
       [1.0000000e+00],
       [9.7790766e-01],
       ...,
       [7.3472845e-16],
       [4.2330548e-03],
       [1.0000000e+00]], dtype=float32)
```