```
In [63]:  #Text Classification on IMDB Dataset

In [19]:  from keras.datasets import imdb

In [20]:  (X_train,y_train),(X_test,y_test)=imdb.load_data(num_words=10000)

In [21]:  (X_train.shape,y_train.shape),(X_test.shape,y_test.shape)

Out[21]:  (((25000,), (25000,)), ((25000,), (25000,)))

In [22]:  X_train[0]
                                        ...

In [23]:  y_train[0]

Out[23]:  1

In [24]:  len(X_train[0])

Out[24]:  218

In [25]:  len(X_train[1])

Out[25]:  189

In [26]:  word_index=imdb.get_word_index()

In [27]:  word_index
                                        ...

In [28]:  reverse_word_index = dict([(key,value) for (value,key) in word_index.items()])
```

In [29]:

```
reverse_word_index
```

```
...
```

In [30]:

```
reverse_word_index[100]
```

Out[30]:

```
'after'
```

In [31]:

```
reverse_word_index.get(100)
```

Out[31]:

```
'after'
```

In [32]:

```
reverse_word_index[200]
```

Out[32]:

```
'may'
```

In [33]:

```
reverse_word_index[300]
```

Out[33]:

```
'later'
```

In [34]:

```
#Example to understand binarty sequence
import numpy as np
X=np.zeros((5,5))
```

In [35]:

```
X
```

Out[35]:

```
array([[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.]])
```

In [36]:

```
X[0,[1,2]]=200
```

In [37]:

```
X
```

Out[37]:

```
array([[  0., 200., 200.,   0.,   0.],
       [  0.,   0.,   0.,   0.,   0.],
       [  0.,   0.,   0.,   0.,   0.],
       [  0.,   0.,   0.,   0.,   0.],
       [  0.,   0.,   0.,   0.,   0.]])
```

In [38]:

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

In [39]:

```
X_train_seq = vectorize_sequences(X_train)
```

In [43]:

```
X_train_seq
```

Out[43]:

```
array([[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.]])
```

In [49]:

```
X_train_seq[0]
```

Out[49]:

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

In [50]:

```
len(X_train_seq)
```

Out[50]:

```
25000
```

In [51]:

```
len(X_train[0])
```

Out[51]:

```
218
```

```
In [52]: len(X_train_seq[1000])
```

```
Out[52]: 10000
```

```
In [53]: len(X_train_seq[100])
```

```
Out[53]: 10000
```

```
In [54]: X_test_seq = vectorize_sequences(X_test)
```

```
In [55]: X_test_seq
```

```
Out[55]: array([[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.]])
```

```
In [56]: len(X_test_seq)
```

```
Out[56]: 25000
```

```
In [57]: len(X_test_seq[1])
```

```
Out[57]: 10000
```

```
In [59]: from keras.models import Sequential
         from keras.layers import Dense
```

```
In [60]: model = Sequential()
         model.add(Dense(128,activation='relu', input_shape=(10000,)))
         model.add(Dense(1,activation='sigmoid'))
```

In [61]:

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

In [62]:

```
history = model.fit(X_train_seq,y_train,validation_data=(X_test_seq,y_test),epochs=10)
```

```
Epoch 1/10
782/782 [==============================] - 26s 26ms/step - loss: 0.3133 - accuracy: 0.8688 - val_loss:
0.2870 - val_accuracy: 0.8796
Epoch 2/10
782/782 [==============================] - 10s 13ms/step - loss: 0.1679 - accuracy: 0.9339 - val_loss:
0.3227 - val_accuracy: 0.8733
Epoch 3/10
782/782 [==============================] - 10s 12ms/step - loss: 0.0918 - accuracy: 0.9654 - val_loss:
0.3923 - val_accuracy: 0.8670
Epoch 4/10
782/782 [==============================] - 10s 12ms/step - loss: 0.0352 - accuracy: 0.9902 - val_loss:
0.5367 - val_accuracy: 0.8638
Epoch 5/10
782/782 [==============================] - 9s 12ms/step - loss: 0.0112 - accuracy: 0.9978 - val_loss: 0.
6436 - val_accuracy: 0.8664
Epoch 6/10
782/782 [==============================] - 9s 12ms/step - loss: 0.0036 - accuracy: 0.9997 - val_loss: 0.
6966 - val_accuracy: 0.8667
Epoch 7/10
782/782 [==============================] - 9s 12ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.
7908 - val_accuracy: 0.8674
Epoch 8/10
782/782 [==============================] - 9s 12ms/step - loss: 4.7579e-04 - accuracy: 1.0000 - val_los
s: 0.8512 - val_accuracy: 0.8673
Epoch 9/10
782/782 [==============================] - 10s 13ms/step - loss: 2.8988e-04 - accuracy: 1.0000 - val_los
s: 0.8932 - val_accuracy: 0.8668
Epoch 10/10
782/782 [==============================] - 10s 13ms/step - loss: 1.9465e-04 - accuracy: 1.0000 - val_los
s: 0.9320 - val_accuracy: 0.8668
```

In [ ]: