```
%tensorflow_version 2.x
import tensorflow
tensorflow.__version__
```

```
'2.4.1'
```

```
import numpy as np
from keras.datasets import imdb
from keras import models
from keras import layers
from keras.utils import to_categorical
```

```
# load the dataset but only keep the top n words, zero the rest

top_words = 10000
(training_data, training_targets), (testing_data, testing_targets) = imdb.load_data(num_words
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:159: Vis
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/datasets/imdb.py:160: Vis
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

```
print('training_data.shape :', training_data.shape)
print('testing_data.shape :', testing_data.shape)
```

```
training_data.shape : (25000,)
testing_data.shape : (25000,)
```

concat training data with test data so as to upgrade data's traing : test ratio from 1:1 to to 4:1. Will split this into training/testing data after few steps

```
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)
```

Now Data Exploration

```
print("Categories:", np.unique(targets))
```

```
Categories: [0 1]
```

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

```
Number of unique words: 9998
```

2 unique categories and 9998 unique words

```
length = [len(i) for i in data]
print("Average length of review :", np.mean(length))
```

```
Average length of review : 234.75892
```

Print the Data

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

print("-------DATA------")
print(data[0])
```

```
Label: 1
-------DATA------
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 2
```

decode the data. Use '#' for words which are unknown

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

```
# this film was just brilliant casting location scenery story direction everyone's reall
```

vectorize with dimesion = 10000. Padding 0 used when size less than 10000

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

Vectorize the data

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

Split the data again into Training and Test data

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

Create Squential NLP Model

```
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"))
```

Double-click (or enter) to edit

```
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 50)                500050    
_________________________________________________________________
dropout_5 (Dropout)          (None, 50)                0         
_________________________________________________________________
dense_6 (Dense)              (None, 50)                2550      
_________________________________________________________________
dropout_6 (Dropout)          (None, 50)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 50)                2550      
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 51        
=================================================================
Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
_________________________________________________________________
```

Double-click (or enter) to edit

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

Double-click (or enter) to edit

```
results = model.fit(
 train_x, train_y,
 epochs= 2,
 batch_size = 500,
 validation_data = (test_x, test_y)
)
```

```
Epoch 1/2
80/80 [==============================] - 5s 55ms/step - loss: 0.5150 - accuracy: 0.7481
Epoch 2/2
80/80 [==============================] - 3s 40ms/step - loss: 0.2116 - accuracy: 0.9194
```

Double-click (or enter) to edit

```
print("Test-Accuracy:", np.mean(results.history["val_accuracy"]))
```

```
Test-Accuracy: 0.8939999938011169
```

Predict

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

print("-------DATA------")
print(test_x[0])
```

```
Label: 1.0
-------DATA------
[0. 1. 1. ... 0. 0. 0.]
```

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

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

```
model.predict(test_x[0:1])
```

```
array([[0.9924487]], dtype=float32)
```

✓ 0s completed at 11:50 PM