DL 2

[29]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
#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) # you␣
↪may take top 10,000
#word frequently used review of movies other are discarded
#consolidating data for EDA Exploratory data analysis (EDA) is used by data␣
↪scientists to analyze and
#investigate data sets and summarize their main characteristics
data = np.concatenate((X_train, X_test), axis=0) # axis 0 is first running␣
↪vertically downwards across
#rows (axis 0), axis 1 is second running horizontally across columns (axis 1),
label = np.concatenate((y_train, y_test), axis=0)
```

[30]:
```
X_train.shape
```

[30]: (25000,)

[31]:
```
X_test.shape
```

[31]: (25000,)

[32]:
```
y_train.shape
```

[32]: (25000,)

[33]:
```
y_test.shape
```

[33]: (25000,)

[34]:
```
print("Review is ",X_train[0]) # series of no converted word to vocabulory␣
↪associated with index
print("Review is ",y_train[0])
```

```
Review is  [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, 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, 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, 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]
Review is  1
```

[61]:
```
vocab=imdb.get_word_index() # Retrieve the word index file mapping words to
 ↪indices
```

[13]:
```
y_train
```

[13]:
```
array([1, 0, 0, …, 0, 1, 0])
```

[36]:
```
y_test
```

[36]:
```
array([0, 1, 1, …, 0, 0, 0])
```

[37]:
```
def vectorize(sequences, dimension = 10000): # We will vectorize every review
 ↪and fill it with zeros
#so that it contains exactly 10,000 numbers.
# Create an all-zero matrix of shape (len(sequences), dimension)
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
   results[i, sequence] = 1
  return results
# Now we split our data into a training and a testing set.
# The training set will contain reviews and the testing set
# # Set a VALIDATION set
```

[38]:
```
test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]
test_x.shape
```

[38]:
```
(10000,)
```

[39]:
```
test_y.shape
```

[39]: (10000,)

[40]:
```
train_x.shape
```

[40]: (40000,)

[41]:
```
train_y.shape
```

[41]: (40000,)

[42]:
```
print("Categories:", np.unique(label))
print("Number of unique words:", len(np.unique(np.hstack(data))))
```

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

[43]:
```
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
```

[44]:
```
print("Label:", label[0])
```

```
Label: 1
```

[45]:
```
print("Label:", label[1])
```

```
Label: 0
```

[46]:
```
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, 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, 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, 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]
```

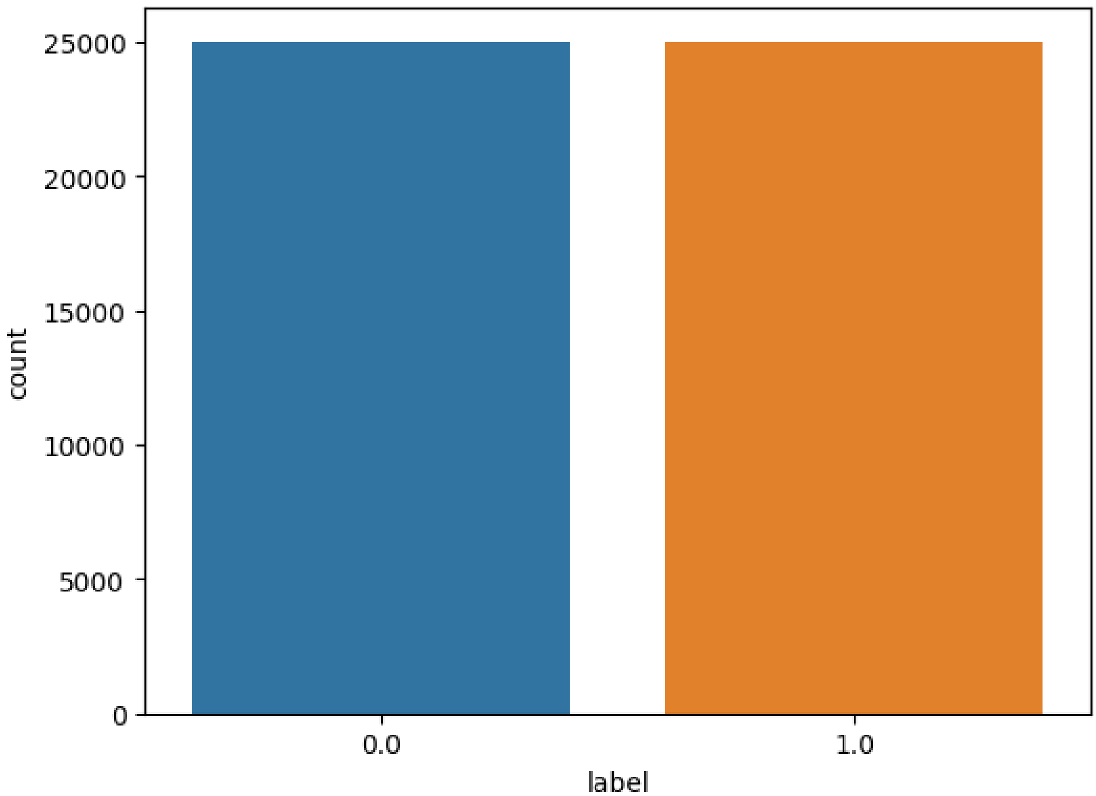
[48]:
```
# Retrieves a dict mapping words to their index in the IMDB dataset.
index = imdb.get_word_index() # word to index
```

```
# Create inverted index from a dictionary with document ids as keys and a list␣
↪of terms as values for
#each document
reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to␣
↪word
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
# The indices are offset by 3 because 0, 1 and 2 are reserved indices for␣
↪"padding", "start of sequence"
#and "unknown".
print(decoded)
```

```
# this film was just brilliant casting location scenery story direction
everyone's really suited 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 loved 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 for
# 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 good and this definitely was also # to the two little
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 that play them all grown
up are such a big profile for the whole film but 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
```

[49]:
```
import seaborn as sns
#Adding sequence to data
# Vectorization is the process of converting textual data into numerical␣
↪vectors and is a process that is
#usually applied once the text is cleaned.
data = vectorize(data)
label = np.array(label).astype("float32")
labelDF=pd.DataFrame({'label':label})
sns.countplot(x='label', data=labelDF)
```

[49]:
```
<AxesSubplot: xlabel='label', ylabel='count'>
```



[50]:
```
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,␣
↪random_state=1)
X_train.shape
```

[50]: (40000, 10000)

[51]:
```
X_test.shape
```

[51]: (10000, 10000)

[54]:
```
# Let's create sequential model
from keras.utils import to_categorical
from keras import models
from keras import layers
model = models.Sequential()
```

[55]:
```
# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our␣
↪reviews are 10,000 integers
#long.
```

```
# The input-layer takes 10,000 as input and outputs it with a shape of 50.
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))
```

[56]:
```
# Hidden - Layers
# Please note you should always use a dropout rate between 20% and 50%. # here␣
↪in our case 0.3 means
#30% dropout we are using dropout to prevent overfitting.
# By the way, if you want you can build a sentiment analysis without LSTMs,␣
↪then you simply need to
#replace it by a flatten layer:
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"))
```

[57]:
```
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
 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
_________________________________________________________________
```

[58]:
```
import tensorflow as tf
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
# We use the "adam" optimizer, an algorithm that changes the weights and biases
#during training.
# We also choose binary-crossentropy as loss classification) and accuracy as␣
↪our evaluation metric.
```

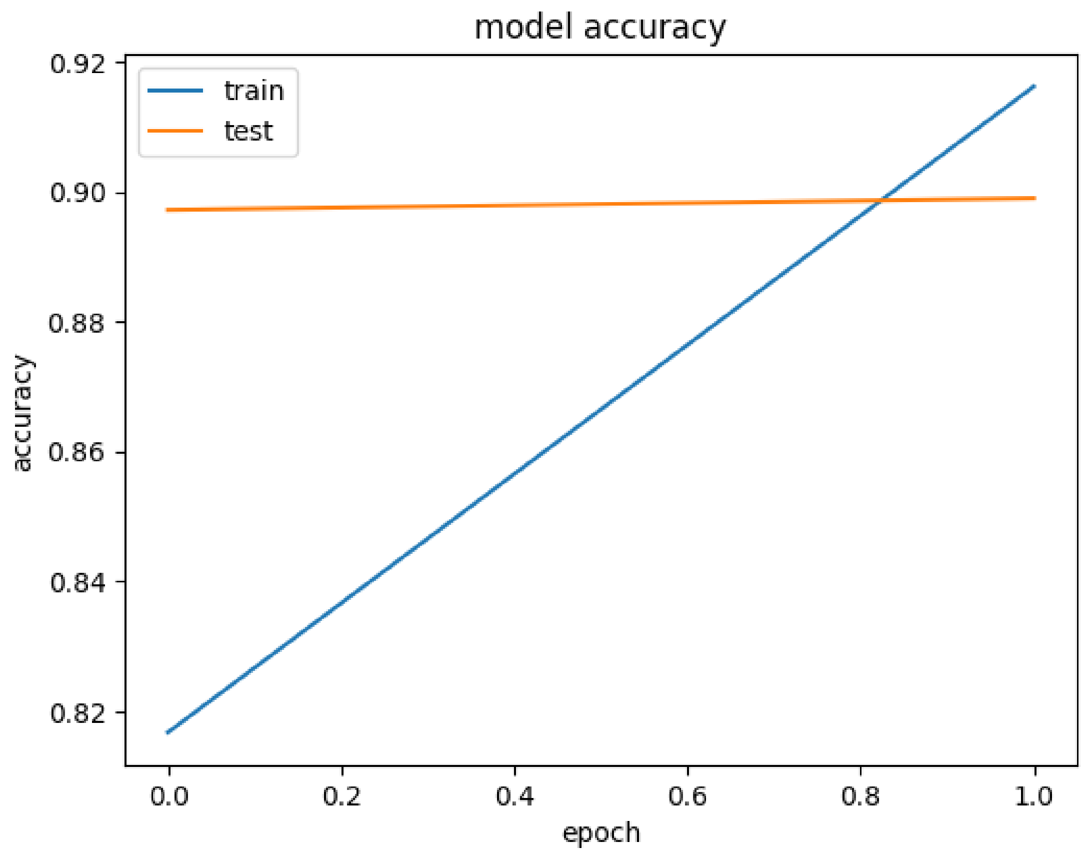
```
model.compile(
optimizer = "adam",
loss = "binary_crossentropy",
metrics = ["accuracy"]
)
from sklearn.model_selection import train_test_split
results = model.fit(
X_train, y_train,
epochs= 2,
batch_size = 500,
validation_data = (X_test, y_test),
callbacks=[callback]
)
# Let's check mean accuracy of our model
print(np.mean(results.history["val_accuracy"]))
# Evaluate the model
score = model.evaluate(X_test, y_test, batch_size=500)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

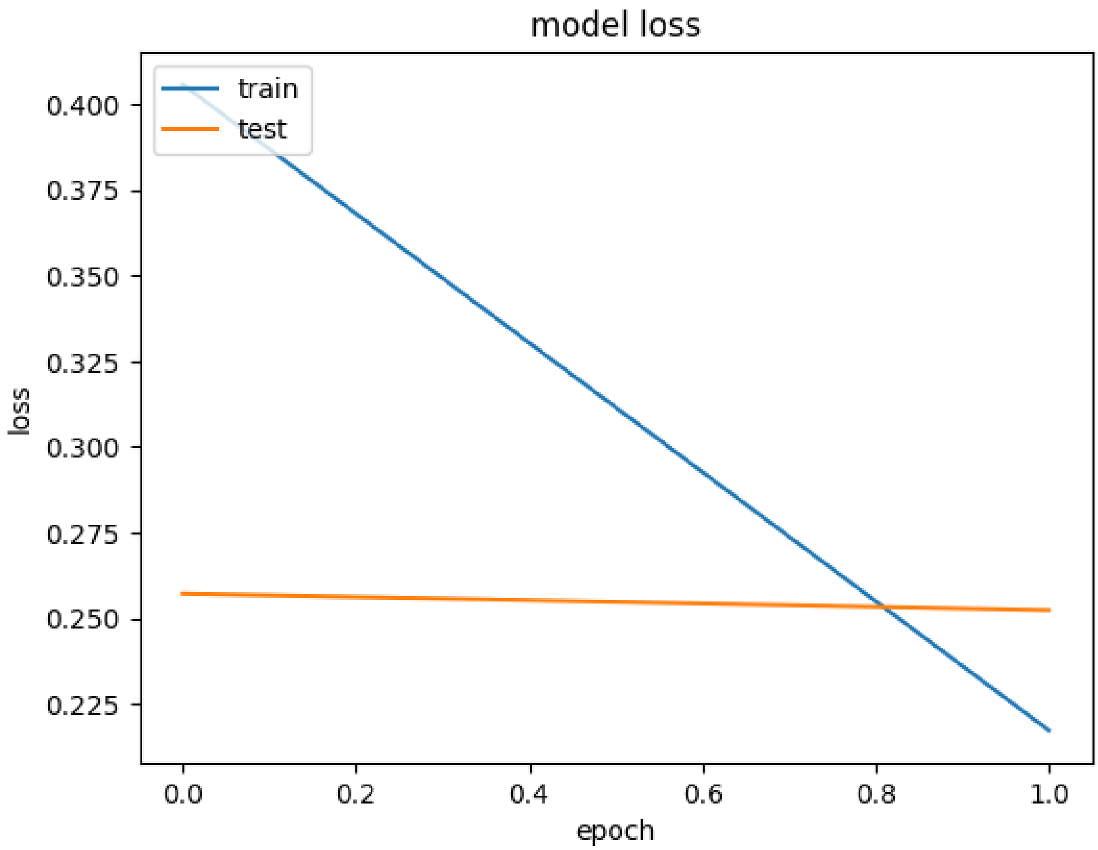
```
Epoch 1/2
80/80 [==============================] - 4s 41ms/step - loss: 0.4057 - accuracy:
0.8167 - val_loss: 0.2572 - val_accuracy: 0.8972
Epoch 2/2
80/80 [==============================] - 2s 31ms/step - loss: 0.2173 - accuracy:
0.9162 - val_loss: 0.2524 - val_accuracy: 0.8990
0.8980999886989594
20/20 [==============================] - 0s 19ms/step - loss: 0.2524 - accuracy:
0.8990
Test loss: 0.25236746668815613
Test accuracy: 0.8989999890327454
```

[60]:
```
#Let's plot training history of our model.
from matplotlib import pyplot as plt
print(results.history.keys())
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()
# list all data in history
# summarize history for accuracy

# 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'])
```




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