IMDB LSTM

In [ ]:

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

In [ ]:

```
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
```

In [ ]:

```
np.random.seed(7)
```

In [ ]:

```
top_words = 20000
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
```

```
WARNING:tensorflow:The `nb_words` argument in `load_data` has been renamed
`num_words`.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-d
atasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
```

In [ ]:

```
print(X_train[1])
print(type(X_train[1]))
print(len(X_train[1]))
```

```
[1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 71
5, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 2
1, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 6
47, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 1
9, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 39
8, 4, 1649, 26, 6853, 5, 163, 11, 3215, 10156, 4, 1153, 9, 194, 775, 7, 82
55, 11596, 349, 2637, 148, 605, 15358, 8003, 15, 123, 125, 68, 2, 6853, 1
5, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 17
4, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4,
9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 6
4, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154,
462, 33, 89, 78, 285, 16, 145, 95]
<class 'list'>
189
```

In [ ]:

```
max_review_length = 80
```

In [ ]:

```
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])
```

```
(25000, 80)
[ 125   68    2 6853   15  349  165 4362   98    5    4  228    9   43
    2 1157   15  299  120    5  120  174   11  220  175  136   50    9
 4373  228 8255    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 1690   15   16    4 1355    5   28    6
   52  154  462   33   89   78  285   16  145   95]
```

Creating Model LSTM 1

In [ ]:

```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_length
))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 80, 32)            640032    
                                                                 
 lstm (LSTM)                 (None, 100)               53200     
                                                                 
 dense (Dense)               (None, 1)                 101       
                                                                 
=================================================================
Total params: 693,333
Trainable params: 693,333
Non-trainable params: 0
_________________________________________________________________
None
```

Fitting the model

In [ ]:

```
model.fit(X_train, y_train, epochs=10, batch_size=64)
```

```
Epoch 1/10
391/391 [==============================] - 48s 115ms/step - loss: 0.4500 -
accuracy: 0.7820
Epoch 2/10
391/391 [==============================] - 45s 115ms/step - loss: 0.2615 -
accuracy: 0.8951
Epoch 3/10
391/391 [==============================] - 45s 115ms/step - loss: 0.1849 -
accuracy: 0.9312
Epoch 4/10
391/391 [==============================] - 45s 116ms/step - loss: 0.1219 -
accuracy: 0.9550
Epoch 5/10
391/391 [==============================] - 45s 116ms/step - loss: 0.0777 -
accuracy: 0.9739
Epoch 6/10
391/391 [==============================] - 49s 125ms/step - loss: 0.0527 -
accuracy: 0.9824
Epoch 7/10
391/391 [==============================] - 50s 128ms/step - loss: 0.0428 -
accuracy: 0.9853
Epoch 8/10
391/391 [==============================] - 48s 123ms/step - loss: 0.0357 -
accuracy: 0.9879
Epoch 9/10
391/391 [==============================] - 47s 119ms/step - loss: 0.0228 -
accuracy: 0.9929
Epoch 10/10
391/391 [==============================] - 45s 115ms/step - loss: 0.0199 -
accuracy: 0.9936
```

Out[ ]:

```
<keras.callbacks.History at 0x7fa5fccca090>
```

In [ ]:

```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 80.10%
```

In [ ]:

```
top_words = 10000
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
```

```
WARNING:tensorflow:The `nb_words` argument in `load_data` has been renamed
`num_words`.
```

In [ ]:

```
print(X_train[1])
print(type(X_train[1]))
print(len(X_train[1]))
```

```
[1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 71
5, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 2
1, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 6
47, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 1
9, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 39
8, 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, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 1
75, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 15
2, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8,
168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 7
8, 285, 16, 145, 95]
<class 'list'>
189
```

In [ ]:

```
max_review_length = 200
```

In [ ]:

```
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])
```

```
(25000, 200)
[   0    0    0    0    0    0    0    0    0    0    0    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  114
    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 1157   15  299  120    5
  120  174   11  220  175  136   50    9 4373  228 8255    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
 1690   15   16    4 1355    5   28    6   52  154  462   33   89   78
  285   16  145   95]
```

Creating LSTM 2

In [ ]:

```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_length
))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 200, 32)           320032    
                                                                 
 lstm_1 (LSTM)               (None, 100)               53200     
                                                                 
 dense_1 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 373,333
Trainable params: 373,333
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
model.fit(X_train, y_train, epochs=10, batch_size=64)
```

```
Epoch 1/10
391/391 [==============================] - 100s 250ms/step - loss: 0.4615
- accuracy: 0.7762
Epoch 2/10
391/391 [==============================] - 97s 249ms/step - loss: 0.2447 -
accuracy: 0.9045
Epoch 3/10
391/391 [==============================] - 97s 248ms/step - loss: 0.1944 -
accuracy: 0.9281
Epoch 4/10
391/391 [==============================] - 97s 248ms/step - loss: 0.1514 -
accuracy: 0.9441
Epoch 5/10
391/391 [==============================] - 97s 248ms/step - loss: 0.1165 -
accuracy: 0.9602
Epoch 6/10
391/391 [==============================] - 98s 250ms/step - loss: 0.0921 -
accuracy: 0.9686
Epoch 7/10
391/391 [==============================] - 97s 247ms/step - loss: 0.0810 -
accuracy: 0.9715
Epoch 8/10
391/391 [==============================] - 97s 249ms/step - loss: 0.0658 -
accuracy: 0.9781
Epoch 9/10
391/391 [==============================] - 98s 251ms/step - loss: 0.0589 -
accuracy: 0.9798
Epoch 10/10
391/391 [==============================] - 98s 251ms/step - loss: 0.0561 -
accuracy: 0.9822
```

Out[ ]:

```
<keras.callbacks.History at 0x7fa604fd8890>
```

In [ ]:

```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 84.78%
```

In [ ]:

```
top_words = 9000
```

In [ ]:

```
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
```

```
WARNING:tensorflow:The `nb_words` argument in `load_data` has been renamed
`num_words`.
```

In [ ]:

```
print(X_train[1])
print(type(X_train[1]))
print(len(X_train[1]))
```

```
[1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 71
5, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 2
1, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 6
47, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 1
9, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 39
8, 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, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 1
75, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 2, 131, 152,
491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 16
8, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78,
285, 16, 145, 95]
<class 'list'>
189
```

In [ ]:

```
max_review_length = 600
```

In [ ]:

```
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])
```

```
(25000, 600)
[   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    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    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    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    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    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    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    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    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    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    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    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    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    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
    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    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    0    0    0    0    0    0
    0    0    0    0    0    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  114    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 1157   15  299  120    5  120  174   11  220  175  136
   50    9 4373  228 8255    5    2  656  245 2350    5    4    2  131
  152  491   18    2   32 7464 1212   14    9    6  371   78   22  625
   64 1382    9    8  168  145   23    4 1690   15   16    4 1355    5
   28    6   52  154  462   33   89   78  285   16  145   95]
```

Creating LSTM 3

In [ ]:

```
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words+1, embedding_vecor_length, input_length=max_review_length
))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 600, 32)           288032    
                                                                 
 lstm_2 (LSTM)               (None, 100)               53200     
                                                                 
 dense_2 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 341,333
Trainable params: 341,333
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:

```
model.fit(X_train, y_train, epochs=10, batch_size=64)
```

```
Epoch 1/10
391/391 [==============================] - 291s 738ms/step - loss: 0.4551
- accuracy: 0.7836
Epoch 2/10
391/391 [==============================] - 284s 727ms/step - loss: 0.2921
- accuracy: 0.8834
Epoch 3/10
391/391 [==============================] - 285s 730ms/step - loss: 0.2102
- accuracy: 0.9218
Epoch 4/10
391/391 [==============================] - 282s 722ms/step - loss: 0.1766
- accuracy: 0.9354
Epoch 5/10
391/391 [==============================] - 281s 720ms/step - loss: 0.1871
- accuracy: 0.9291
Epoch 6/10
391/391 [==============================] - 282s 720ms/step - loss: 0.1476
- accuracy: 0.9458
Epoch 7/10
391/391 [==============================] - 280s 717ms/step - loss: 0.1462
- accuracy: 0.9458
Epoch 8/10
391/391 [==============================] - 282s 721ms/step - loss: 0.1059
- accuracy: 0.9637
Epoch 9/10
391/391 [==============================] - 281s 719ms/step - loss: 0.1159
- accuracy: 0.9603
Epoch 10/10
391/391 [==============================] - 283s 724ms/step - loss: 0.0814
- accuracy: 0.9736
```

Out[ ]:

```
<keras.callbacks.History at 0x7fa601abbed0>
```

In [ ]:

```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 85.73%
```