IMDB - Embedding with LSTM

> ## NLP(Natural Language Processing)

```
import warnings
warnings.filterwarnings('ignore')
```

Import Keras

- Keras Version 확인

```
import keras

keras.__version__
```

```
'2.4.3'
```

I. IMDB Data_Set Load & Review

1) Load IMDB Data_Set

- Word to Vector
- 전체 데이터 내에서 단어의 사용빈도에 따라 인덱스화
- 정수 인덱스 '11'은 11번째로 자주 사용된 단어를 나타냄
- num_words = 10000 : 인덱스 값 10000 이하의 단어만 추출
- 단어 인덱스 값이 10000을 넘지 않는 단어만 분석에 사용

```
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
17465344/17464789 [==============================] - 0s 0us/step
```

2) Visualization & Frequency(Optional)

- x - Histogram(리뷰 길이)

```
import matplotlib.pyplot as plt

print('리뷰 최대 길이 :', max(len(L) for L in X_train))
print('리뷰 평균 길이 :', sum(map(len, X_train))/len(X_train))

plt.figure(figsize = (9, 6))
plt.hist([len(L) for L in X_train], bins = 50)
plt.xlabel('Length of X_train')
plt.ylabel('Number of X_train')
plt.show()
```

- y - Frequency(0:부정, 1:긍정)

```
import numpy as np

unique_elements, counts_elements = np.unique(y_train, return_counts = True)

print('Label 빈도수:')
print(np.asarray((unique_elements, counts_elements)))
```

```
Label 빈도수:
[[    0     1]
 [12500 12500]]
```

II. Tensor Transformation

1) X_train & X_test : (25000, 10000)

- vectorization
  - (25000, 10000)

```
from keras import preprocessing

X_train = preprocessing.sequence.pad_sequences(X_train, maxlen = 10000)
X_test = preprocessing.sequence.pad_sequences(X_test, maxlen = 10000)

X_train.shape, X_test.shape
```

```
((25000, 10000), (25000, 10000))
```

- Transformation Check

```
print(X_train[0][:21])
print(X_train[0][9979:])
```

```
print(X_test[0][:21])
print(X_test[0][9979:])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[ 226   65   16   38 1334   88   12   16  283    5   16 4472  113  103
   32   15   16 5345   19  178   32]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[  14  286  170    8  157   46    5   27  239   16  179    2   38   32
   25 7944  451  202   14    6  717]
```

2) y_train & y_test

```
y_train = np.asarray(y_train).astype(float)
y_test = np.asarray(y_test).astype(float)

print(y_train[:21])
print(y_test[:21])
```

```
[1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0.]
[0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1.]
```

III. Keras Embedding Modeling

1) Model Define

- 모델 신경망 구조 정의
  - Embedding Dimension : 32

```
from keras import models
from keras import layers

imdb = models.Sequential()

imdb.add(layers.Embedding(10000, 32, input_length = 10000))

imdb.add(layers.LSTM(8))
imdb.add(layers.Dropout(0.5))
imdb.add(layers.Dense(1, activation = 'sigmoid'))
```

- 모델 구조 확인

```
imdb.summary()
```

```
Model: "sequential"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 10000, 32)         320000
_________________________________________________________________
lstm (LSTM)                  (None, 8)                 1312
_________________________________________________________________
dropout (Dropout)            (None, 8)                 0
_________________________________________________________________
dense (Dense)                (None, 1)                 9
=================================================================
Total params: 321,321
Trainable params: 321,321
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

- 모델 학습방법 설정

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

3) Model Fit

- 약 20분

```
%%time

Hist_imdb = imdb.fit(X_train, y_train,
                     epochs = 25,
                     batch_size = 512,
                     validation_data = (X_test, y_test))
```

```
Epoch 1/25
49/49 [==============================] - 73s 823ms/step - loss: 0.6897 - accuracy: 0.5539 - v
Epoch 2/25
49/49 [==============================] - 39s 808ms/step - loss: 0.5980 - accuracy: 0.7335 - v
Epoch 3/25
49/49 [==============================] - 39s 807ms/step - loss: 0.4749 - accuracy: 0.8261 - v
Epoch 4/25
49/49 [==============================] - 40s 813ms/step - loss: 0.4549 - accuracy: 0.8256 - v
Epoch 5/25
49/49 [==============================] - 40s 813ms/step - loss: 0.3831 - accuracy: 0.8697 - v
Epoch 6/25
49/49 [==============================] - 39s 803ms/step - loss: 0.3564 - accuracy: 0.8837 - v
Epoch 7/25
49/49 [==============================] - 40s 813ms/step - loss: 0.3254 - accuracy: 0.8964 - v
Epoch 8/25
```

```
49/49 [==============================] - 39s 805ms/step - loss: 0.3120 - accuracy: 0.8999 - v
Epoch 9/25
49/49 [==============================] - 39s 810ms/step - loss: 0.3362 - accuracy: 0.8821 - v
Epoch 10/25
49/49 [==============================] - 39s 804ms/step - loss: 0.2807 - accuracy: 0.9137 - v
Epoch 11/25
49/49 [==============================] - 39s 805ms/step - loss: 0.2603 - accuracy: 0.9258 - v
Epoch 12/25
49/49 [==============================] - 39s 811ms/step - loss: 0.2401 - accuracy: 0.9328 - v
Epoch 13/25
49/49 [==============================] - 39s 808ms/step - loss: 0.2782 - accuracy: 0.9106 - v
Epoch 14/25
49/49 [==============================] - 39s 803ms/step - loss: 0.2546 - accuracy: 0.9235 - v
Epoch 15/25
49/49 [==============================] - 39s 802ms/step - loss: 0.2230 - accuracy: 0.9359 - v
Epoch 16/25
49/49 [==============================] - 39s 806ms/step - loss: 0.2090 - accuracy: 0.9414 - v
Epoch 17/25
49/49 [==============================] - 39s 809ms/step - loss: 0.1984 - accuracy: 0.9450 - v
Epoch 18/25
49/49 [==============================] - 39s 807ms/step - loss: 0.1935 - accuracy: 0.9486 - v
Epoch 19/25
49/49 [==============================] - 39s 809ms/step - loss: 0.1810 - accuracy: 0.9528 - v
Epoch 20/25
49/49 [==============================] - 39s 805ms/step - loss: 0.1746 - accuracy: 0.9563 - v
Epoch 21/25
49/49 [==============================] - 39s 810ms/step - loss: 0.1689 - accuracy: 0.9562 - v
Epoch 22/25
49/49 [==============================] - 39s 808ms/step - loss: 0.1617 - accuracy: 0.9594 - v
Epoch 23/25
49/49 [==============================] - 40s 812ms/step - loss: 0.1496 - accuracy: 0.9644 - v
Epoch 24/25
49/49 [==============================] - 39s 810ms/step - loss: 0.1483 - accuracy: 0.9656 - v
Epoch 25/25
49/49 [==============================] - 39s 806ms/step - loss: 0.1399 - accuracy: 0.9666 - v
CPU times: user 15min 18s, sys: 27.9 s, total: 15min 46s
Wall time: 16min 57s
```

4) 학습 결과 시각화

- Loss Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_imdb.history['loss']) + 1)

plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_imdb.history['loss'])
plt.plot(epochs, Hist_imdb.history['val_loss'])
plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.grid()
```

```
plt.show()
```

- Accuracy Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_imdb.history['accuracy']) + 1)

plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_imdb.history['accuracy'])
plt.plot(epochs, Hist_imdb.history['val_accuracy'])
plt.title('Training & Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.grid()
plt.show()
```

5) Model Evaluate

- Loss & Accuracy

```
loss, accuracy = imdb.evaluate(X_test, y_test)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))
```

```
782/782 [==============================] - 89s 114ms/step - loss: 0.4533 - accuracy: 0.8475
Loss = 0.45331
Accuracy = 0.84748
```

#

#

#

The End

#

#

#