IMDB - Embedding with DNN

> ## 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.Flatten())

imdb.add(layers.Dense(16))
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    
_________________________________________________________________
flatten (Flatten)            (None, 320000)            0         
_________________________________________________________________
dense (Dense)                (None, 16)                5120016   
_________________________________________________________________
dropout (Dropout)            (None, 16)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17        
=================================================================
Total params: 5,440,033
Trainable params: 5,440,033
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 = 50,
                     batch_size = 512,
                     validation_data = (X_test, y_test))
```

```
Epoch 23/50
49/49 [==============================] - 20s 408ms/step - loss: 0.0042 - accuracy: 1.0000
Epoch 24/50
49/49 [==============================] - 20s 407ms/step - loss: 0.0035 - accuracy: 1.0000
Epoch 25/50
49/49 [==============================] - 20s 407ms/step - loss: 0.0035 - accuracy: 1.0000
Epoch 26/50
49/49 [==============================] - 20s 409ms/step - loss: 0.0032 - accuracy: 1.0000
Epoch 27/50
49/49 [==============================] - 20s 407ms/step - loss: 0.0027 - accuracy: 1.0000
Epoch 28/50
49/49 [==============================] - 20s 406ms/step - loss: 0.0028 - accuracy: 1.0000
Epoch 29/50
```

```
49/49 [==============================] - 20s 407ms/step - loss: 0.0021 - accuracy: 1.0000
Epoch 30/50
49/49 [==============================] - 20s 408ms/step - loss: 0.0022 - accuracy: 0.9999
Epoch 31/50
49/49 [==============================] - 20s 406ms/step - loss: 0.0019 - accuracy: 1.0000
Epoch 32/50
49/49 [==============================] - 20s 406ms/step - loss: 0.0018 - accuracy: 1.0000
Epoch 33/50
49/49 [==============================] - 20s 406ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 34/50
49/49 [==============================] - 20s 408ms/step - loss: 0.0018 - accuracy: 1.0000
Epoch 35/50
49/49 [==============================] - 20s 409ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 36/50
49/49 [==============================] - 20s 409ms/step - loss: 0.0015 - accuracy: 1.0000

Epoch 37/50
49/49 [==============================] - 20s 409ms/step - loss: 0.0014 - accuracy: 1.0000
Epoch 38/50
49/49 [==============================] - 20s 409ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 39/50
49/49 [==============================] - 20s 414ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 40/50
49/49 [==============================] - 20s 405ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 41/50
49/49 [==============================] - 20s 406ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 42/50
49/49 [==============================] - 20s 409ms/step - loss: 9.3620e-04 - accuracy: 1.0
Epoch 43/50
49/49 [==============================] - 20s 405ms/step - loss: 0.0010 - accuracy: 1.0000
Epoch 44/50
49/49 [==============================] - 20s 407ms/step - loss: 9.9379e-04 - accuracy: 1.0
Epoch 45/50
49/49 [==============================] - 20s 405ms/step - loss: 9.0731e-04 - accuracy: 1.0
Epoch 46/50
49/49 [==============================] - 20s 408ms/step - loss: 9.1410e-04 - accuracy: 1.0
Epoch 47/50
49/49 [==============================] - 20s 406ms/step - loss: 8.2664e-04 - accuracy: 1.0
Epoch 48/50
49/49 [==============================] - 20s 406ms/step - loss: 7.7742e-04 - accuracy: 1.0
Epoch 49/50
49/49 [==============================] - 20s 406ms/step - loss: 7.1441e-04 - accuracy: 1.0
Epoch 50/50
49/49 [==============================] - 20s 407ms/step - loss: 7.4424e-04 - accuracy: 1.0
CPU times: user 16min 1s, sys: 25 s, total: 16min 26s
Wall time: 16min 37s
```

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 [==============================] - 8s 10ms/step - loss: 0.6794 - accuracy: 0.8753
Loss = 0.67936
Accuracy = 0.87528
```

#

#

#

The End

#

```
#

#
```