IMDB - Binary Classification

> NLP(Natural Language Processing)

Import Tensorflow

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

- TensorFlow '1.x' Version 지정

```
# %tensorflow_version 1.x
import tensorflow as tf

tf.__version__
```

```
'2.6.0'
```

- GPU 설정 확인

```
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

- GPU 정보 확인

```
!nvidia-smi
```

```
Wed Sep 29 02:22:19 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   43C    P0    57W / 149W |    122MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

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 tensorflow.keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
```

```
max(max(W) for W in train_data)
```

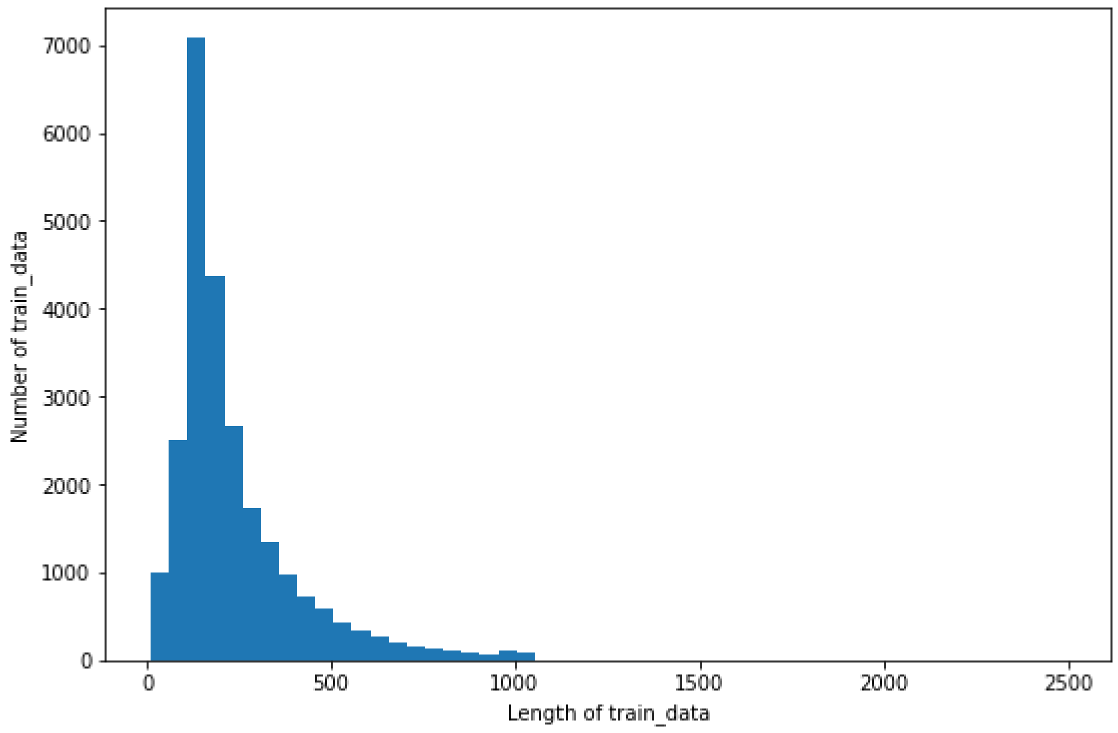
```
9999
```

2) Visualization & Frequency(Optional)

- x - Histogram(리뷰 길이)

```
import matplotlib.pyplot as plt

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

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

```
리뷰 최대 길이 : 2494
리뷰 평균 길이 : 238.71364
```



- y - Frequency(0:부정, 1:긍정)

```
import numpy as np

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

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

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

3) Data Structure Review(Optional)

```
# 전체 train_data 개수
print(len(train_data))

# 첫번째 train_data 정보
print(len(train_data[0]))
print(train_data[0][0:10])
```

```
print(train_data[0].count(4))
print(train_labels[0])
```

```
25000
218
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
15
1
```

4) Vector to Word(Optional)

- get_word_index( ) : 단어와 인덱스를 매핑한 사전
- 0, 1, 2 : '패딩', '문서 시작', '사전에 없음'

```
word_index = imdb.get_word_index()

print(word_index)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders': 16115, 'hanging': 2345, 'woody'
```

- 인덱스와 단어 위치 변경

```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

print(reverse_word_index)
```

```
{34701: 'fawn', 52006: 'tsukino', 52007: 'nunnery', 16816: 'sonja', 63951: 'vani', 1408: 'woods', 16115: 'spiders', 2345: 'hanging', 2289: '
```

- 0번 영화 리뷰 디코딩(1:긍정)

```
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])

print(decoded_review)
print(train_labels[0])
```

```
? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just ima
1
```

- 1번 영화리뷰 디코딩(0:부정)

```
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[1]])

print(decoded_review)
print(train_labels[1])
```

```
? big hair big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horror movies a
0
```

II. Tensor Transformation

1) X_train & X_test : (25000, 10000)

- vectorize_sequence( ) 정의
- 크기는 10000이고 모든 원소가 0인 행렬 생성
  - np.zeros(len(sequences), dimension))

- 값이 존재하는 인덱스의 위치를 1로 지정
    - enumerate( )
    - results[i, sequence] = 1.0

```
import numpy as np

def vectorize_sequences(sequences, dimension = 10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.0
    return results
```

- enumerate( ) - Example

```
r = np.zeros((5, 10))
v = [1, 3, 5, 7, 9]

for i, v in enumerate(v):
    r[i, v] = 1.0

print(r)
```

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

- vectorize_sequence( ) 적용

```
X_train = vectorize_sequences(train_data)
X_test = vectorize_sequences(test_data)

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

2) y_train & y_test

```
y_train = np.asarray(train_labels).astype(float)
y_test = np.asarray(test_labels).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.]
```

3) Train vs. Validation Split

```
X_valid = X_train[:10000]
partial_X_train = X_train[10000:]

y_valid = y_train[:10000]
partial_y_train = y_train[10000:]

partial_X_train.shape, partial_y_train.shape, X_valid.shape, y_valid.shape
```

```
((15000, 10000), (15000,), (10000, 10000), (10000,))
```

III. IMDB Keras Modeling

1) Model Define

- 모델 신경망 구조 정의

```
from tensorflow.keras import models
from tensorflow.keras import layers

imdb = models.Sequential()
imdb.add(layers.Dense(16, activation = 'relu', input_shape = (10000,)))
imdb.add(layers.Dense(16, activation = 'relu'))
imdb.add(layers.Dense(1, activation = 'sigmoid'))
```

- 모델 구조 확인

```
imdb.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                160016
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

- 모델 학습방법 설정

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

3) Model Fit

- 약 1분

```
%%time

Hist_imdb = imdb.fit(partial_X_train, partial_y_train,
                     epochs = 50,
                     batch_size = 512,
                     validation_data = (X_valid, y_valid))
```

```
30/30 [==============================] - 1s 21ms/step - loss: 0.0017 - accuracy: 0.9999 - val_loss: 0.7625 - val_accuracy: 0.8646
Epoch 23/50
30/30 [==============================] - 1s 20ms/step - loss: 0.0068 - accuracy: 0.9980 - val_loss: 0.8004 - val_accuracy: 0.8665
Epoch 24/50
30/30 [==============================] - 1s 20ms/step - loss: 9.9355e-04 - accuracy: 1.0000 - val_loss: 0.8200 - val_accuracy: 0.8647
Epoch 25/50
30/30 [==============================] - 1s 21ms/step - loss: 8.3293e-04 - accuracy: 1.0000 - val_loss: 0.8575 - val_accuracy: 0.8650
Epoch 26/50
30/30 [==============================] - 1s 20ms/step - loss: 0.0044 - accuracy: 0.9990 - val_loss: 0.8950 - val_accuracy: 0.8634
Epoch 27/50
30/30 [==============================] - 1s 20ms/step - loss: 4.6545e-04 - accuracy: 1.0000 - val_loss: 0.9161 - val_accuracy: 0.8622
Epoch 28/50
30/30 [==============================] - 1s 22ms/step - loss: 3.8997e-04 - accuracy: 1.0000 - val_loss: 0.9505 - val_accuracy: 0.8610
Epoch 29/50
30/30 [==============================] - 1s 19ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 0.9856 - val_accuracy: 0.8620
Epoch 30/50
30/30 [==============================] - 1s 21ms/step - loss: 2.3789e-04 - accuracy: 1.0000 - val_loss: 1.0044 - val_accuracy: 0.8617
Epoch 31/50
30/30 [==============================] - 1s 21ms/step - loss: 1.9726e-04 - accuracy: 1.0000 - val_loss: 1.0424 - val_accuracy: 0.8593
Epoch 32/50
30/30 [==============================] - 1s 20ms/step - loss: 0.0042 - accuracy: 0.9989 - val_loss: 1.0794 - val_accuracy: 0.8601
Epoch 33/50
30/30 [==============================] - 1s 20ms/step - loss: 1.3371e-04 - accuracy: 1.0000 - val_loss: 1.0951 - val_accuracy: 0.8600
Epoch 34/50
30/30 [==============================] - 1s 21ms/step - loss: 1.0621e-04 - accuracy: 1.0000 - val_loss: 1.1120 - val_accuracy: 0.8594
Epoch 35/50
30/30 [==============================] - 1s 21ms/step - loss: 8.6088e-05 - accuracy: 1.0000 - val_loss: 1.1521 - val_accuracy: 0.8593
Epoch 36/50
30/30 [==============================] - 1s 20ms/step - loss: 0.0016 - accuracy: 0.9993 - val_loss: 1.2354 - val_accuracy: 0.8611
Epoch 37/50
30/30 [==============================] - 1s 21ms/step - loss: 5.1585e-05 - accuracy: 1.0000 - val_loss: 1.2205 - val_accuracy: 0.8589
Epoch 38/50
30/30 [==============================] - 1s 21ms/step - loss: 3.9013e-05 - accuracy: 1.0000 - val_loss: 1.2386 - val_accuracy: 0.8589
Epoch 39/50
30/30 [==============================] - 1s 20ms/step - loss: 3.2772e-05 - accuracy: 1.0000 - val_loss: 1.2750 - val_accuracy: 0.8595
Epoch 40/50
30/30 [==============================] - 1s 20ms/step - loss: 2.5375e-05 - accuracy: 1.0000 - val_loss: 1.3379 - val_accuracy: 0.8577
Epoch 41/50
30/30 [==============================] - 1s 21ms/step - loss: 0.0045 - accuracy: 0.9988 - val_loss: 1.3804 - val_accuracy: 0.8576
Epoch 42/50
30/30 [==============================] - 1s 20ms/step - loss: 1.8001e-05 - accuracy: 1.0000 - val_loss: 1.3845 - val_accuracy: 0.8578
Epoch 43/50
30/30 [==============================] - 1s 20ms/step - loss: 1.4431e-05 - accuracy: 1.0000 - val_loss: 1.3936 - val_accuracy: 0.8595
Epoch 44/50
30/30 [==============================] - 1s 22ms/step - loss: 1.2003e-05 - accuracy: 1.0000 - val_loss: 1.4080 - val_accuracy: 0.8589
Epoch 45/50
30/30 [==============================] - 1s 20ms/step - loss: 9.8698e-06 - accuracy: 1.0000 - val_loss: 1.4356 - val_accuracy: 0.8589
Epoch 46/50
30/30 [==============================] - 1s 20ms/step - loss: 9.7232e-04 - accuracy: 0.9997 - val_loss: 1.4856 - val_accuracy: 0.8600
Epoch 47/50
30/30 [==============================] - 1s 21ms/step - loss: 6.7901e-06 - accuracy: 1.0000 - val_loss: 1.4765 - val_accuracy: 0.8597
Epoch 48/50
30/30 [==============================] - 1s 20ms/step - loss: 5.6047e-06 - accuracy: 1.0000 - val_loss: 1.4843 - val_accuracy: 0.8602
Epoch 49/50
30/30 [==============================] - 1s 20ms/step - loss: 5.1077e-06 - accuracy: 1.0000 - val_loss: 1.4981 - val_accuracy: 0.8598
Epoch 50/50
30/30 [==============================] - 1s 21ms/step - loss: 4.4317e-06 - accuracy: 1.0000 - val_loss: 1.5242 - val_accuracy: 0.8592
CPU times: user 46.5 s, sys: 2.73 s, total: 49.2 s
Wall time: 42.6 s
```

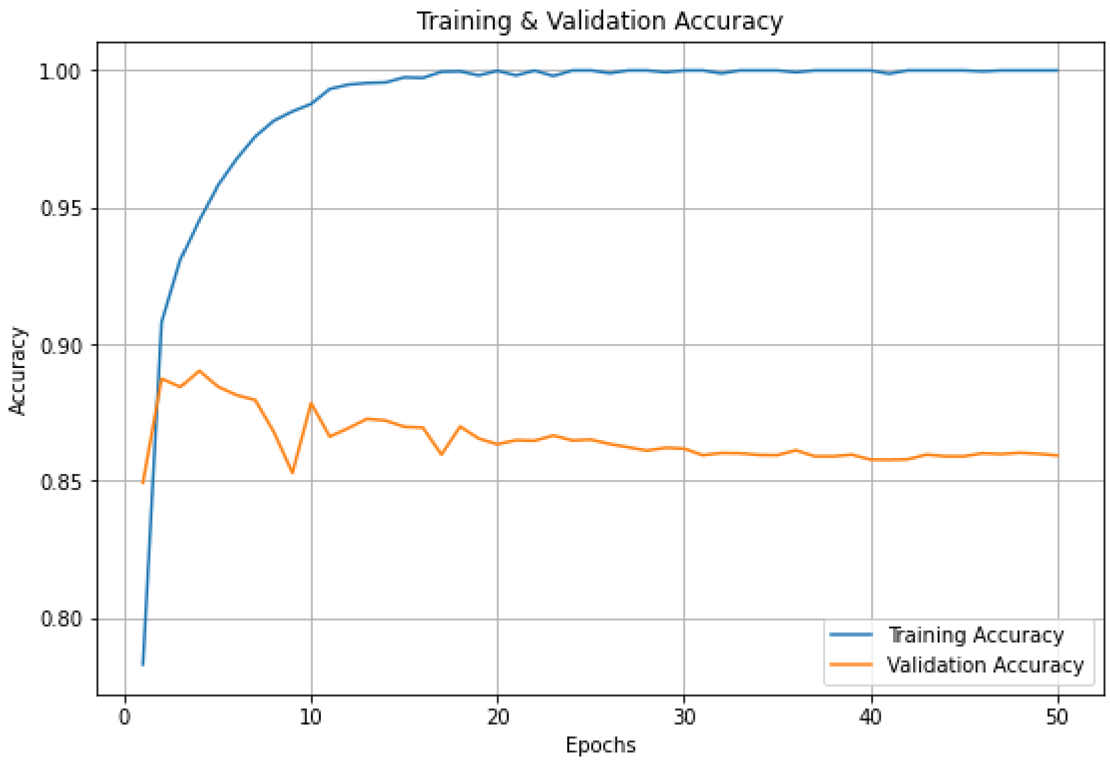
4) 학습 결과 시각화

- 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 [==============================] - 2s 3ms/step - loss: 1.6677 - accuracy: 0.8432
Loss = 1.66768
Accuracy = 0.84316
```

6) Model Predict

```
np.argmax(imdb.predict(X_test))
```

1

#

#

#

The End

#

#

#

1초 오전 11:51에 완료됨