Chapter5. 텍스트 유사도

Kaggle의 <Quora Question Paris> 데이터를 사용하였다. Quora는 질문과 답변을 할 수 있는 사이트로 많은 질문 중에는 중복된 것들이 포함되기 때문에 이를 걸러내면 좋을 것이다. 이미 Quora에서는 랜덤 포레스트 모델을 이용하여 중복에 대한 검사를 하고 있다.

1. 데이터 전처리

```
import numpy as np
import pandas as pd
import re
import json
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
```

```
#중복인 경우와 중복이 아닌 경우로 데이터를 나눈 후 중복이 아닌 개수가 비슷하도록 데이터의 일부를 다시 뽑는다.
train_pos_data=train_data.loc[train_data['is_duplicate']==1]
train_neg_data=train_data.loc[train_data['is_duplicate']==0]

class_difference=len(train_neg_data)-len(train_pos_data)
sample_frac=1-(class_difference/len(train_neg_data))
train_neg_data=train_neg_data.sample(frac=sample_frac)
```

```
print('중복 질문 개수: {}'.format(len(train_pos_data)))
print('중복이 아닌 질문 개수:{}'.format(len(train_neg_data)))
```

```
중복 질문 개수: 149263
중복이 아닌 질문 개수:149263
```

```
#라벨에 따라 나눠진 데이터를 다시 하나로 합친다.
train_data=pd.concat([train_neg_data, train_pos_data])
```

```
#토크나이징 , 각 단어를 인덱스로 바꾸기
tokenizer=Tokenizer()
tokenizer.fit_on_texts(filtered_questions1+filtered_questions2)
questions1_sequence=tokenizer.texts_to_sequences(filtered_questions1)
questions2_sequence=tokenizer.texts_to_sequences(filtered_questions2)
#전체 데이터의 길이를 맞추기 위해 정의한 최대 길이보다 긴 문장은 자르고 짧은 문장은 패딩하기
MAX_SEQUENCE_LENGTH=31
q1_data=pad_sequences(questions1_sequence, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
q2_data=pad_sequences(questions2_sequence, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
```

```
#물음표와 마침표 같은 구두점 및 기호를 제거하고 모든 문자를 소문자로 바꾼다.
FILTERS='[^a-zA-Z]'
change_filter=re.compile(FILTERS)
questions1=[str(s) for s in train_data['question1']]
questions2=[str(s) for s in train_data['question2']]

filtered_questions1=list()
filtered_questions2=list()


for q in questions1:
  filtered_questions1.append(re.sub(change_filter, "",q).lower())
for q in questions2:
  filtered_questions2.append(re.sub(change_filter,"",q).lower())
```

```
#데이터 크기 확인
word_vocab={}
word_vocab=tokenizer.word_index
word_vocab["<PAD>"]=0

labels=np.array(train_data['is_duplicate'], dtype=int)

print('Shape of question1 data: {}'.format(q1_data.shape))
print('Shape of question2 data: {}'.format(q2_data.shape))
print('Shape of label: {}'.format(labels.shape))
print('Words in index: {}'.format(len(word_vocab)))
```

```
Shape of question1 data: (298526, 31)
Shape of question2 data: (298526, 31)
Shape of label: (298526,)
Words in index: 387392
```

```
#단어 사전과 전체 단어의 개수는 딕셔너리 형태로 저장
data_configs={}
data_configs['vocab']=word_vocab
data_configs['vocab_size']=len(word_vocab)
```

```
TRAIN_Q1_DATA='q1_train.npy'
TRAIN_Q2_DATA='q2_train.npy'
TRAIN_LABEL_DATA='label_train.npy'
DATA_CONFIGS='data_configs.json'

np.save(open(DATA_IN_PATH+TRAIN_Q1_DATA, 'wb'), q1_data)
np.save(open(DATA_IN_PATH+TRAIN_Q2_DATA, 'wb'), q2_data)
np.save(open(DATA_IN_PATH+TRAIN_LABEL_DATA, 'wb'), labels)
json.dump(data_configs, open(DATA_IN_PATH+DATA_CONFIGS, 'w'))
```

Test data도 위와 같은 방법대로 전처리한다.

2. XG부스트(eXtream Gradient Boosting)

앙상블(여러 개의 학습 알고리즘을 사용해 더 좋은 성능을 얻는 방법)의 한 방법인 부스팅 기법을 사용한다. 부스팅은 각 결과를 이전 알고리즘, 모델이 학습 후 잘못 예측한 부분에 가중치를 둬서 다시 모델로 가서 학습하는 방식으로 결과를 순차적으로 취합한다. 여러 개의 학습 알고리즘, 모델을 통해 각각 결과를 예측하고 모든 결과를 동등하게 보고 취합해서 결과를 얻는 방식은 배깅하고는 다르다. 최종적으로 XG부스트는 트리 부스팅 방식에 경사 하강법을 통해 최적화하는 방법이다. 또한 연산량을 줄이기 위해 의사결정 트리를 구성할 때 병렬 처리를 사용해 빠른 시간에 학습이 가능하다.

```
#데이터를 하나씩 묶어 하나의 질문 쌍으로 만든다.
train_input=np.stack((train_q1_data, train_q2_data), axis=1)
```

모델 생성&학습

```
from sklearn.model_selection import train_test_split
train_input, eval_input, train_label, eval_label=train_test_split(train_input, train_labels, test_size=0.2, random_state=4242)
```

```
import xgboost as xgb

train_data=xgb.DMatrix(train_input.sum(axis=1), label=train_label)
eval_data=xgb.DMatrix(eval_input.sum(axis=1), label=eval_label)

data_list=[(train_data, 'train'), (eval_data, 'valid')]
```

```
params={}
#목적 함수 : 이진 로지스틱 함수
params['objective']='binary:logistic'
#평가 지표 : rmse
params['eval_metric']='rmse'

bst=xgb.train(params, train_data, num_boost_round=1000, evals=data_list, early_stopping_rounds=18)
```

(877 스텝에서 멈춘 책과 다르게 53 스텝에서 멈추었다.)

Predict 적용

```
test_input=np.stack((test_q1_data, test_q2_data), axis=1)
test_data=xgb.DMatrix(test_input.sum(axis=1))
test_predict=bst.predict(test_data)
```

```
import pandas as pd
output=pd.DataFrame({'test_id':test_id_data, 'is_duplicate':test_predict})
output.to_csv(DATA_IN_PATH+'simple_xgb.csv', index=False)
```

kaggle에 제출한 결과 이렇게 나왔다.

Name	Submitted	Wait time	Execution time	Score
simple_xgb.csv	a minute ago	1 seconds	30 seconds	2.76285

Complete

3. CNN 텍스트 유사도 분석 모델

모델 구현

```
#문장에 대한 정보를 하나의 벡터로 만든다.
import tensorflow as tf
import tensorflow.keras.layers as layers
class SentenceEmbedding(layers.Layer):
  def __init__(self, **kargs):
    super(SentenceEmbedding, self).__init__()
    self.conv=layers.Conv1D(kargs['conv_num_filters'], kargs['conv_window_size'],
                            activation=tf.keras.activations.relu, padding='same')
    self.max_pool=layers.MaxPool1D(kargs['max_pool_seq_len'], 1)
    #특징값의 차원수 조절
    self.dense=layers.Dense(kargs['sent_embedding_dimension'], activation=tf.keras.activations.relu)
  def call(self, x):
    x=self.conv(x)
    x=self.max_pool(x)
    x=self.dense(x)
    return tf.squeeze(x, 1)
```

```
class SentenceSimilarityModel(tf.keras.Model):
  def __init__(self, **kargs):
    super(SentenceSimilarityModel, self).__init__(name=kargs['model_name'])
    self.word_embedding=layers.Embedding(kargs['vocab_size'],
                                         kargs['word_embedding_dimension'])
    self.base_encoder=SentenceEmbedding(**kargs)
    self.hype_encoder=SentenceEmbedding(**kargs)
    self.dense=layers.Dense(kargs['hidden_dimension'],
                            activation=tf.keras.activations.relu)
    self.logit=layers.Dense(1, activation=tf.keras.activations.sigmoid) #출력값 0~1
    self.dropout=layers.Dropout(kargs['dropout_rate'])
  def call(self, x):
    x1, x2=x
    b_x=self.word_embedding(x1)
    h_x=self.word_embedding(x2)
    b_x=self.dropout(b_x)
    h_x=self.dropout(h_x)
    b_x=self.base_encoder(b_x)
    h_x=self.hype_encoder(h_x)
    e_x=tf.concat([b_x, h_x], -1)
    e_x=self.dense(e_x)
    e_x=self.dropout(e_x)
    return self.logit(e_x)
```

모델 하라미터 정의

```
model_name='cnn_similarity'
BATCH_SIZE=1024
NUM_EPOCHS=100
VALID_SPLIT=0.1
MAX_LEN=31
#vocab_size, word_embedding_dimension : 단어 임베딩을 위한 차원 값
#conv_num_filters, conv_window_size : 합성곱 레이어를 위한 차원값과 윈도우의 크기
#max_pool_seq_len : 맥스 풀링을 위한 고정 길이, 합성곱 레이어에서 활용
#sent_embedding_dimension : 문장 임베딩에 대한 차원 값
#hidden dimension : 마지막 dense 레이어에 대한 차원 값
kargs={'model_name':model_name, 'vocab_size':prepo_configs['vocab_size'],
       'word_embedding_dimension':100, 'conv_num_filters':300,
       'conv_window_size':3, 'max_pool_seq_len':MAX_LEN,
       'sent_embedding_dimension':128, 'dropout_rate':0.2,
       'hidden_dimension':200, 'output_dimension':1}
```

모델 생성

```
import os
earlystop_callback=tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=1)
#min delta : the threshold that triggers the termination
#patience : no improvement epochs
checkpoint_path=DATA_IN_PATH+'weights.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)

cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True,
                              save_weights_only=True)
```

모델 학습

```
import os
earlystop_callback=tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=1)
#min delta : the threshold that triggers the termination
#patience : no improvement epochs
checkpoint_path=DATA_IN_PATH+'weights.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)

cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True,
                            save_weights_only=True)
```

```
history=model.fit((q1_data, q2_data), labels, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
                  validation_split=VALID_SPLIT, callbacks=[earlystop_callback, cp_callback])
```

```
Epoch 1/100
263/263 [==============================] - 310s 1s/step - loss: 0.1403 - accuracy: 0.9429 - val_loss: 0.5520 - val_accuracy: 0.7354

Epoch 00001: val_accuracy improved from -inf to 0.73537, saving model to /content/drive/MyDrive/Kaggle/quora-question-pairs/weights.h5
Epoch 2/100
263/263 [==============================] - 310s 1s/step - loss: 0.0104 - accuracy: 0.9969 - val_loss: 0.8640 - val_accuracy: 0.6948

Epoch 00002: val_accuracy did not improve from 0.73537
```

Kaggle에 제출한 결과 아래와 같다.

Name	Submitted	Wait time	Execution time	Score
cnn_predict.csv	a minute ago	1 seconds	49 seconds	1.80673

Complete

4. MaLSTM(Manhattan Distance+LSTM)

일반적으로 문장의 유사도를 계산할 때 코사인 유사도를 사용하는데 이 모델은 맨해튼 거리를 사용한다.

모델 구현 & 생성 & 학습

```
import tensorflow as tf
class Model(tf.keras.Model):
  def __init__(self, **kargs):
    super(Model, self).__init__(name=model_name)
    #임베딩 층
    self.embedding=tf.keras.layers.Embedding(input_dim=kargs['vocab_size'],
                                             output_dim=kargs['embedding_dimension'])
    #LSTM 층
    self.lstm=tf.keras.layers.LSTM(units=kargs['lstm_dimension'])
  def call(self, x):
    x1, x2=x
    x1=self.embedding(x1)
    x2=self.embedding(x2)
    x1=self.lstm(x1)
    x2=self.lstm(x2)
    #거리가 멀수록 유사도 값은 작아지고 거리가 가까울수록 유사도는 1에 가까워짐.
    x=tf.exp(-tf.reduce_sum(tf.abs(x1-x2), axis=1))
    return x
```

```
model_name='malstm_similarity'
BATCH_SIZE=128
NUM_EPOCHS=5
VALID_SPLIT=0.1

kargs={'vocab_size':prepro_configs['vocab_size'],
       'embedding_dimension':100, 'lstm_dimension':150,}
model=Model(**kargs)
model.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

```
import os
earlystop_callback=tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=1)
#min delta : the threshold that triggers the termination
#patience : no improvement epochs
checkpoint_path=DATA_IN_PATH+'malstm_weights.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)

cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True,
                            save_weights_only=True)
```

```
history=model.fit((q1_data, q2_data), labels, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
                  validation_split=VALID_SPLIT, callbacks=[earlystop_callback, cp_callback])
```

```
Epoch 1/5
2100/2100 [==============================] - 1284s 609ms/step - loss: 0.8208 - accuracy: 0.5202 - val_loss: 0.6887 - val_accuracy: 0.5522

Epoch 00001: val_accuracy improved from -inf to 0.55218, saving model to /content/drive/MyDrive/Kaggle/quora-question-pairs/malstm_weights.h5
Epoch 2/5
2100/2100 [==============================] - 1248s 594ms/step - loss: 0.4417 - accuracy: 0.8574 - val_loss: 0.6545 - val_accuracy: 0.6397

Epoch 00002: val_accuracy improved from 0.55218 to 0.63973, saving model to /content/drive/MyDrive/Kaggle/quora-question-pairs/malstm_weights.h5
Epoch 3/5
2100/2100 [==============================] - 1229s 585ms/step - loss: 0.2913 - accuracy: 0.9434 - val_loss: 0.2828 - val_accuracy: 0.8865

Epoch 00003: val_accuracy improved from 0.63973 to 0.88649, saving model to /content/drive/MyDrive/Kaggle/quora-question-pairs/malstm_weights.h5
Epoch 4/5
2100/2100 [==============================] - 1221s 581ms/step - loss: 0.1208 - accuracy: 0.9726 - val_loss: 0.3189 - val_accuracy: 0.9091

Epoch 00004: val_accuracy improved from 0.88649 to 0.90906, saving model to /content/drive/MyDrive/Kaggle/quora-question-pairs/malstm_weights.h5
Epoch 5/5
2100/2100 [==============================] - 1236s 589ms/step - loss: 0.0914 - accuracy: 0.9821 - val_loss: 0.3065 - val_accuracy: 0.9168

Epoch 00005: val_accuracy improved from 0.90906 to 0.91684, saving model to /content/drive/MyDrive/Kaggle/quora-question-pairs/malstm_weights.h5
```

Kaggle에 제출한 결과 아래와 같다.

Name	Submitted	Wait time	Execution time	Score
rnn_predict.csv	a few seconds ago	1 seconds	9 seconds	18.73791

Complete