코딩과제 6 – 마케팅 접목을 위한 RNN 모델 개발 및 BERT 모델 조사

0-1. 조원 소개

20140269 고혁훈 20160768 김홍엽 20160886 이규복 20180623 조성은

0-2. 개요

1-1. NLP task 소개 및 task 선정

1-2. Q&A 마케팅 활용방안

2-1. simpleRNN이용한 챗봇 구현 from scratch

2-2. LSTM-seq2seq을 이용한 챗봇 구현 from scratch

3-1. Bert 소개

3-2. API를 통한 kobert Q&A 구현

4. NLP 최신 모델 동향

1-1-1 NLP task 소개 – 감성분석, 의미분석, 구문분석, 질의응답

NLP(Natural Language Processing)의 세부 분야로는 감성분석, 의미분석, 구문분석, 음성인식(질의응답) 등이 있다. 감성분석은 텍스트에 포함된 의견이나 감정, 평가, 태도 등의 주관적인 정보를 컴퓨터를 이용해 분석하는 것이다. 감성분석은 두 가지의 아키텍쳐로 이루어져 있다. 첫째는 Opinion Definition으로, 문장의 어떤 부분에 의견이 있는지를 정의내리는 것이다. 이를 찾는 과정은 개체, 감성, 주제, 발화시점의 4가지 요소를 정의 내림으로써 시작한다. 둘째는 Opinion Summerization으로 첫번째 과정에서 분석한 의견을 요약하는 것이다. 다음으로, 의미 분석은 문장이 옳은 문장인지 아닌지를 분석하는 것이다. 예를 들어, "말이 많다"라는 문장이 있을 때, 여기서 말이 horse인지, talking을 의미하는 것인지를 분석하는 것이다. 구문분석은 문장이 구조적으로 옳은 문장인지 아닌지를 분석하는 것이다. 마지막으로 음성인식(질의응답)은 사용자의 질문에 대한 답변이 될 수 있는 응답을 데이터베이스 내에서 탐색하여 사용자에게 제시하는 것이다. 질의응답 시스템은 사용자의 질의에 관련된 데이터를 검색하는 후보검색 단계와 검색된 데이터에서 정답을 생성하는 정답추출 단계로 구성된다. 질의응답 시스템이 작동하는 과정을 조금 더 상세히 알아보면, 처음에 자연어로 된 질문이 컴퓨터에 들어오고, 이 질문이 컴퓨터에 입력된다. 컴퓨터는 단어사전을 이용해 구문해석을 하고, 의미사전을 이용해 의미를 해석하는 과정을 거친다. 이후, 데이터베이스를 이용해 질문의 답이 될 수 있는 데이터를 찾아내고 결과를 생성한다. 마지막으로 응답문을 생성한 뒤 이를 출력한다

출처:"What is Chatbot Marketing? -Guide", sendpulse, ,https://sendpulse.com/support/glossary/chatbot-marketing, (2021.11.16)

"Question Answering System", aistudy, http://www.aistudy.co.kr/linguistics/natural/question_answer_system.htm, (2021.11.16)

1-1-2 NLP task 선정 – 질의응답

이렇게 인공지능 속의 NLP에는 다양한 분야가 있지만 이번 과제에서는 질의응답 모델을 집중적으로 다루어보고자 한다. 질의응답 모델은 챗봇에 특화된 모델로서 다음으로는 이 모델이 챗봇에 사용되어 마케팅에 어떻게 활용될 수 있는지와 그 현황에 대해 알아본다.

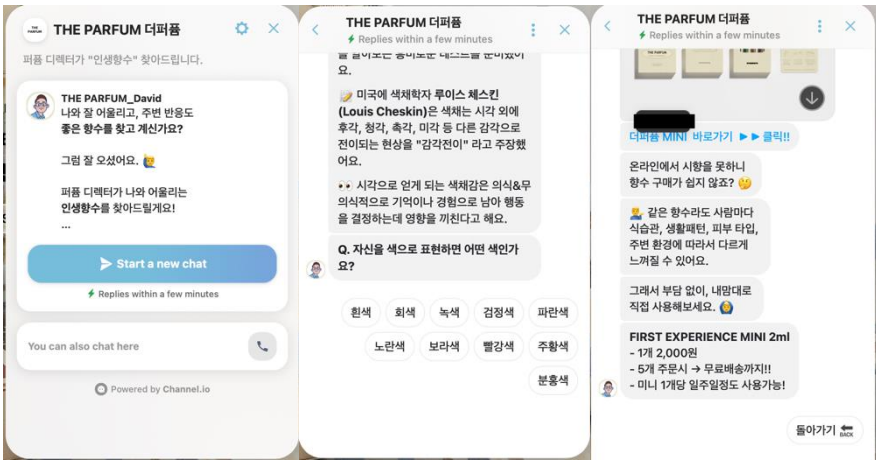
챗봇이란 "음성이나 문자를 통한 인간과의 대화에서 특정한 작업을 수행하도록 제작된 컴퓨터 프로그램"이다. 챗봇은 언어이해 방식, 검색 방식, 각본 방식의 총 세가지 방식으로 구현될 수 있다. 언어이해 방식은 자연어를 이해해서 대화를 진행하는 방식으로 NLP를 가장 고도화된 수준으로 사용한다. 검색 방식은 입력 받은 말에서 특정 단어나 어구를 추출하여 그에 맞는 미리 준비된 응답을 출력하는 방식이다. 마지막으로 각본 방식은 미리 각본을 만든 후, 사용자의 입력에 대한 동작과 각본에 있는 응답을 출력하는 방식이다.

1-2 Q&A – 챗봇 마케팅 활용방안

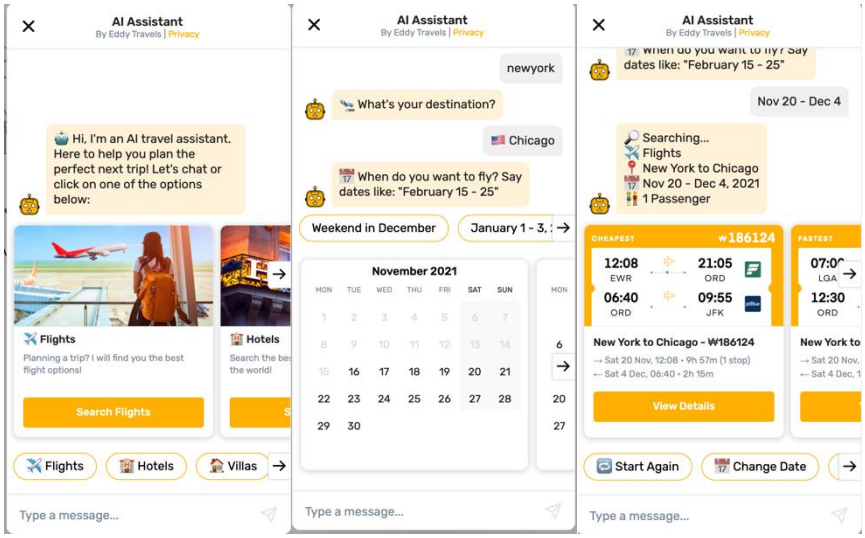
현재 대부분의 기업은 고객의 간단한 문의나 질의에 대해 답변하는 상담 효율화를 목적으로 챗봇을 도입하는 경우가 많지만 챗봇을 마케팅에 활용하는 사례도 점차 증가하고 있는 현황이다. 따라서 이번 과제의 목적에 맞게 챗봇을 마케팅에 활용하는 사례를 구체적으로 살펴보고자 한다. 챗봇마케팅이란 세일을 증가시키려는 목적으로 고객과 자동적으로 소통하게 하는 컴퓨터 프로그램을 도입시키는 것이다.

한국 스타트업 기업인 더 퍼퓸(theparfum.co.kr)은 챗봇을 마케팅에 활용하고 있다. 아래에는 팀원 중 한 명이 직접 경험한 챗봇 대화의 장면으로, 챗봇이 묻는 여러 질문에 대한 2~4개의 선택지 중 하나의 답변들을 선택한 결과, 챗봇이 개인에게 가장 잘 어울릴 것 같은 향수를 추천해주고 있다. 이를 통해, 챗봇과의 대화의 결론을 잠재적 고객의 자사 제품 구매로 이어지게 하는 해당 기업의 마케팅 전략을 경험해 볼 수 있었다.

출처: "국내 챗봇 성공 사례 6가지 + 챗봇이 필요한 이유", 채널톡, Blaire, 2020.09.10, https://channel.io/ko/blog/best-chatbot-case-6, ,



다음으로는, eddytravels(eddytravels.com)에서 사용하고 있는 챗봇 마케팅을 소개하고자 한다. Eddytravels는 고객이 챗봇과의 대화를 통해 flights, hotels, tours and activities, apartments, weekend flights, apply for visa, travel insurance, delayed baggage protection, airport shuttle까지 총 9개의 제품 및 서비스를 선택하게 하고 있다. 고객은 챗봇과의 대화를 통해 빠르고 쉽게 제품 및 서비스를 선택 및 구매가 가능하다. 아래는 filghts를 예매한다고 가정한 한 팀원이 직접 챗봇 서비스를 이용한 장면이다.



결론적으로, 챗봇을 이용한 기업의 마케팅 방식은 기업에게 더 적은 비용과 시간으로 고객 응대를 가능하게 하고, 고객은 더 빠르고 쉬운 방법으로 기업에게 자신이 원하는 바를 전달할 수 있게 되어 기업은 더 많은 트래픽의 창출, 증가한 세일과 고객 경험 향상으로 이어지는 장점을 누릴 수 있음을 알 수 있었다.

이후의 보고서에서는 질의응답 방식의 NLP(챗봇과 Q&A)를 직접 코딩 해보고 그 결과와 인사이트를 도출한 뒤, NLP에 대해 조금 더 상세히 논의해본도록 한다.

출처: "감정 분석(Sentiment Analysis)", Data Science by Yngie, 2020.07.31, https://yngie-c.github.io/nlp/2020/07/31/sentiment_analysis/,

2. LSTM을 이용한 챗봇 구현 from scratch with tensorflow

2-1. simpleRNN

2-1-1 데이터 준비

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
from tensorflow.keras.utils import to_categorical
import pandas as pd

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN
chatbotData=pd.read_csv("ChatData.csv")
question, answer = list(chatbotData["Q"]) , list(chatbotData["A"])
print(len(question))
print(len(answer))
total_len = 1000

text = ''

for i in range(total_len) :
    text = text + question[i] +' '
    text = text + answer[i] +'\n'
text=text.replace('.','').replace('!','').replace('?','').replace(',','')
```

2-1-2 전처리

```
t = Tokenizer()
t.fit_on_texts([text])
vocab_size = len(t.word_index) + 1

print('단어 집합의 크기 : %d' % vocab_size)
sequences = list()
for line in text.split('\n'): # Wn을 기준으로 문장 토큰화
    encoded = t.texts_to_sequences([line])[0]
    for i in range(1, len(encoded)):
        sequence = encoded[:i+1]
        sequences.append(sequence)

print('학습에 사용할 샘플의 개수: %d' % len(sequences))
```

2-1-3 모델링

```
max_len = max(len(l) for l in sequences)
```

```
sequences = pad_sequences(sequences, maxlen=max_len, padding='pre')

sequences = np.array(sequences)
X = sequences[:, :-1]
y = sequences[:, -1]
y = to_categorical(y, num_classes=vocab_size)
model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=max_len-
1)) # 임베딩 벡터는 10 차원, 레이블을 분리하였으므로 이제 X 의 길이는 5
model.add(SimpleRNN(32))
model.add(Dense(vocab_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
'])
model.fit(X, y, epochs=100, verbose=1)
```

```
Epoch 97/100
5754/5754 [==============================] - 1s 193us/sample - loss: 1.0579 -
accuracy: 0.7946
Epoch 98/100
5754/5754 [==============================] - 1s 193us/sample - loss: 1.0385 -
accuracy: 0.8015
```

2-1-4 예측 및 예시

```
def sentence_generation(model, t, current_word, n): # 모델, 토크나이저, 현재 단어, 반복
할 횟수
    init_word = current_word # 처음 들어온 단어도 마지막에 같이 출력하기위해 저장
    sentence = ''
    for _ in range(n): # n 번 반복
        encoded = t.texts_to_sequences([current_word])[0] # 현재 단어에 대한 정수 인코
딩
        encoded = pad_sequences([encoded], maxlen=max_len-
1, padding='pre') # 데이터에 대한 패딩
        result = model.predict_classes(encoded, verbose=0)
    # 입력한 X(현재 단어)에 대해서 Y 를 예측하고 Y(예측한 단어)를 result 에 저장.
        for word, index in t.word_index.items():
            if index == result: # 만약 예측한 단어와 인덱스와 동일한 단어가 있다면
                break # 해당 단어가 예측 단어이므로 break
        current_word = current_word + ' '  + word # 현재 단어 + ' ' + 예측 단어를 현재
 단어로 변경
        sentence = sentence + ' ' + word # 예측 단어를 문장에 저장
    # for 문이므로 이 행동을 다시 반복
    sentence = 'com: '+ sentence
    return sentence

def chat_start():
    while True:
        q = input('\nyou : ')
        if q == '종료' :
            print('채팅을 종료합니다')
```

```
            break
        else:
            print(sentence_generation(model, t, q, 3))
```

```
chat_start()
```

you → input

```
you : 누구니
com:  저는 저는 위로봇입니다

you : 치킨 기프티콘 받았어
com:  부러워요 잘 해보세요

you : 너무 많이 먹었어
com:  소화제 드세요 뭐가

you : 가족이랑 여행 가려고
com:  좋은 생각이에요 해요

you : 여자친구랑 가도 될까
com:  휴식 중 부담스러워하지

you : 종료
채팅을 종료합니다
```

2-2. LSTM – seq2seq

2-2-1 데이터 준비

```
chatbotData=pd.read_csv("/home/hyukhun/chatbots/ChatData.csv")
question, answer = list(chatbotData["Q"]) , list(chatbotData["A"])
```

2-2-2 데이터 가공

pos morph 분석 + split + 특수문자 제거(RE_FILTER = re.compile("[.,!?₩':;~()]'")) + 원핫인코딩

```
xEncoder = convertTextToIndex(question, wordToIndex, ENCODER_INPUT)
xDecoder = convertTextToIndex(answer, wordToIndex, DECODER_INPUT)
yDecoder = convertTextToIndex(answer, wordToIndex, DECODER_TARGET)
```

```
[0] : [11288  9241  8527     0     0     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 2980  954 2846 10312 3334    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0]

[0] : [ 2980   954  2846 10312  3334     2     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0]
```

2-2-3 모델링 및 학습

```
#훈련 모델 생성
#인코더 정의
encoderInputs=layers.Input(shape=(None,))
encoderOutputs=layers.Embedding(len(words),embeddingDim)(encoderInputs)
encoderOutputs,stateH, stateC=layers.LSTM(lstmHiddenDim,return_state=True,
            dropout=0.2, recurrent_dropout=0.5)(encoderOutputs)
```

```
encoderStates=[stateH, stateC]
#디코더 정의
decoderInputs=layers.Input(shape=(None,))
decoderEmbedding=layers.Embedding(len(words),
                                  embeddingDim)
decoderOutputs=decoderEmbedding(decoderInputs)
decoderLSTM=layers.LSTM(lstmHiddenDim,
                        return_state=True,
            return_sequences=True,
                        dropout=0.2,
                        recurrent_dropout=0.5)
decoderOutputs, _, _=decoderLSTM(decoderOutputs,initial_state=encoderStates)
decoderDense=layers.Dense(len(words),
                          activation="softmax")
decoderOutputs=decoderDense(decoderOutputs)
model=models.Model([encoderInputs, decoderInputs],
             decoderOutputs)
```

학습

```
model.compile(optimizer='adam',
             loss='categorical_crossentropy',
             metrics=['accuracy'])
for epoch in range(7):
    print("total epoch:", epoch+1)
    history=model.fit([xEncoder,xDecoder], yDecoder,
             epochs=100,
             batch_size=64,
                     verbose=0)
    print("accuracy :", history.history['accuracy'])
    print("loss :", history.history['loss'])
    #문장 예측
    #3 박 4 일 놀러 가고 싶다 -> 여행 은 언제나 좋죠

    inputEncoder=xEncoder[2].reshape(1,xEncoder[2].shape[0]) #(30,) ->(1,30)
    inputDecoder=xDecoder[2].reshape(1,xDecoder[2].shape[0]) #(30,) ->(1,30)

    results=model.predict([inputEncoder,inputDecoder])

    #결과값에 대해서 가장 큰 값의 위치를 구함
    index=np.argmax(results[0], 1)
    #인덱스 -> 문장으로 변환
    sentence=convertIndexToText(index, indexToWord)
    print(sentence)
model.save_weights('./checkpoints/my_checkpoint')
```

2-2-4 테스트

```
Q : 3박 4일 놀러가고 싶다
A : 여행은 언제나 좋죠.
```

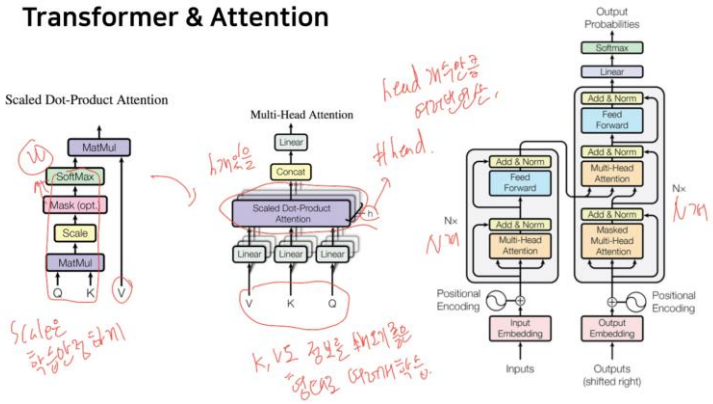
```
idx = 4
model.load_weights('./checkpoints/my_checkpoint')
inputEncoder=xEncoder[idx].reshape(1,xEncoder[idx].shape[0]) #(30,) ->(1,30)
print(" ".join([indexToWord[seg] for seg in xEncoder[idx] if seg!=0]))
inputDecoder=xDecoder[idx].reshape(1,xDecoder[idx].shape[0]) #(30,) ->(1,30)
results=model.predict([inputEncoder,inputDecoder])

#결과값에 대해서 가장 큰 값의 위치를 구함
index=np.argmax(results[0], 1)
#인덱스 -> 문장으로 변환
sentence=convertIndexToText(index, indexToWord)
print(sentence)
```

```
PPL 심하네
2021-11-16 18:50:21.395279: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR
Optimization Passes are enabled (registered 2)
2021-11-16 18:50:21.396311: I
tensorflow/core/platform/profile_utils/cpu_utils.cc:114] CPU Frequency: 3099995000
Hz
눈살 이 찌푸려지죠 .
```

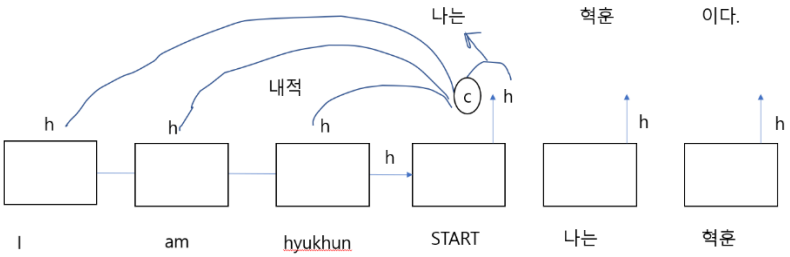
```
질문:  SD 카드 망가졌어
2021-11-16 18:52:24.069045: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR
Optimization Passes are enabled (registered 2)
2021-11-16 18:52:24.070093: I
tensorflow/core/platform/profile_utils/cpu_utils.cc:114] CPU Frequency: 3099995000
Hz
답변:  다시 새로 사는 게 마음 편해요 .
```

3-1. Bert 소개



from transformer papers

BERT는 transformer의 encoder를 차용한 모델이다. BERT는 자연어 처리에 수많은 진보를 가져왔으며 여전히 NLP task의 주류 모델이다. 거기에 쓰인 핵심 기법은 attention이다.



위의 그림처럼 attention은 하나의 hidden layer에 의존하지 않고 이전 히든 레이어들도 같이 참조해서 현재 hidden weight와 이전의 hidden layer의 내적을 통해 유사도를 구한 후 가장 자신과 비슷한 것을 찾는 알고리즘이다. 쉽게 말해 이전 레이들에게 query를 날려서 제일 비슷한 것을 찾도록 만든 것이고 이것을 여러 개 중첩한 것이 transformer이다. 이렇게 attention 네트워크로 구성할 경우, 위치정보를 잃어버리기에 거기에 positional encoding기법을 도입해서 sin함수와 cos함수의 조합으로 유니크한 위치정보를 추가해 준다.

BERT의 핵심은 self-supervised 학습 기법 중 NSP와 MLM을 도입했다는 점이고 특히 MLM을 통해 진정한 bidirectional을 구현해냈다. 이 중 NSP를 제거하고 더 많은 데이터로 추가 학습한 ROBERTa 모델은 현재에도 널리 쓰이고 있다. 그래서 우리는 bert를 이용한 챗봇을 구현해볼 것이고, 처음부터 학습하기에는 resource가 많이 부족해서 대기업에서 미리 학습한 kobert를 불러와서 구현할 것이다. 즉, hugging face API와 korquad를 통해서 쉽게 구현해볼 것이다.

3-2. API를 통한 kobert Q&A 구현

3-2-1 패키지 설치 및 pretrained 불러오기

```
!pip install transformers==2.9.1
!pip install torch
!pip install sentencepiece
!git clone https://github.com/monologg/KoBERT-KorQuAD.git

from transformers import BertModel
from tokenization_kobert import KoBertTokenizer

model = BertModel.from_pretrained('monologg/kobert')
tokenizer = KoBertTokenizer.from_pretrained('monologg/kobert')
```

3-2-2 학습

```
!python3 run_squad.py --model_type kobert \
                      --model_name_or_path monologg/kobert \
                      --output_dir models \
                      --data_dir data \
                      --train_file KorQuAD_v1.0_train.json \
                      --predict_file KorQuAD_v1.0_dev.json \
```

```
                        --evaluate_during_training \
                        --per_gpu_train_batch_size 8 \
                        --per_gpu_eval_batch_size 8 \
                        --max_seq_length 512 \
                        --logging_steps 4000 \
                        --save_steps 4000 \
                        --do_train
```

결과의 일부

```
Evaluating: 100% 828/828 [07:27<00:00,  1.85it/s]
11/15/2021 17:47:43 - INFO - __main__ -     Evaluation done in total 447.176976
secs (0.067529 sec per example)
11/15/2021 17:47:43 - INFO - transformers.data.metrics.squad_metrics -   Writing
predictions to: models/predictions_.json
11/15/2021 17:47:43 - INFO - transformers.data.metrics.squad_metrics -   Writing
nbest to: models/nbest_predictions_.json
11/15/2021 17:48:00 - INFO - __main__ -     HasAns_exact = 39.851056459993075
11/15/2021 17:48:00 - INFO - __main__ -     HasAns_f1 = 48.482574858621554
11/15/2021 17:48:00 - INFO - __main__ -     HasAns_total = 5774
11/15/2021 17:48:00 - INFO - __main__ -     best_exact = 39.851056459993075
11/15/2021 17:48:00 - INFO - __main__ -     best_exact_thresh = 0.0
11/15/2021 17:48:00 - INFO - __main__ -     best_f1 = 48.482574858621554
11/15/2021 17:48:00 - INFO - __main__ -     best_f1_thresh = 0.0
11/15/2021 17:48:00 - INFO - __main__ -     exact = 39.851056459993075
11/15/2021 17:48:00 - INFO - __main__ -     f1 = 48.482574858621554
11/15/2021 17:48:00 - INFO - __main__ -     total = 5774
11/15/2021 17:48:00 - INFO - __main__ -   ***** Official Eval results *****
11/15/2021 17:48:01 - INFO - __main__ -     exact_match = 39.9722895739522
11/15/2021 17:48:01 - INFO - __main__ -     f1 = 64.601108039328
```

3-2-3 평가

```
!python3 evaluate_v1_0.py ./data/KorQuAD_v1.0_dev.json ./models/predictions_.json
```

```
{"exact_match": 47.506061655697955, "f1": 73.62206816223296}
```

3-2-4 테스트 – context를 주고 이름을 물어보기

```
from transformers import BertForQuestionAnswering
from tokenization_kobert import KoBertTokenizer
import torch
MODEL_PATH = "./models"
model = BertForQuestionAnswering.from_pretrained(MODEL_PATH)
tokenizer = KoBertTokenizer.from_pretrained(MODEL_PATH)
```

```
question = input("질문을 입력하세요: ")
```

```
질문을 입력하세요: 당신의 이름은 무엇입니까?
```

```
context = input("문단을 입력하세요: ")
```
```
문단을 입력하세요: 내 이름은 김홍엽입니다. 나이는 25살입니다.
```

```
# 입력 정보 토크나이징 및 임베딩 진행
tokenized_question = tokenizer._tokenize(question)
tokenized_context = tokenizer._tokenize(context)

question_tokens = [tokenizer._convert_token_to_id(q) for q in tokenized_question]
context_tokens = [tokenizer._convert_token_to_id(c) for c in tokenized_context]

token_embeddings = tokenizer.build_inputs_with_special_tokens(question_tokens, cont
ext_tokens)
segment_embeddings = tokenizer.create_token_type_ids_from_sequences(question_tokens
, context_tokens)
tokens = [tokenizer._convert_id_to_token(t) for t in token_embeddings]
```

```
# QA 모델 예측 및 정답 출력
outputs = model(input_ids = torch.tensor([token_embeddings]), token_type_ids=torch.
tensor([segment_embeddings]))
start_index = torch.argmax(outputs[0])
end_index =  torch.argmax(outputs[1])

answer = ' '.join(tokens[start_index:end_index+1])
answer = answer.replace('_', '') # '_'(띄어쓰기) 없애기
print("정답은 \"{}\" 입니다.".format(answer))
```

```
정답은 "김 홍 엽" 입니다.
```

4. NLP 최신 모델 동향 -

최근 응용되는 NLP 모델들은 BERT기반이나 GPT-3 기반이다. Transformer의 encoder 대표격 bert와 decoder의 대표격인 GPT-3가 주류이다. Encoder분야는 BERT에서 XLNET, 그리고 XLNET에서 T5등이 나왔지만 roberta의 등장으로 결국은 BERT로 귀결되었다. 또한 성능개선의 측면은 transformer기반의 아키텍쳐를 중심으로 삼아 더 많은 파라미터와 더 많은 학습데이터로 성능을 끌어올리는 현실이다. 최근 openAI에서 GPT-4를 준비 중에 있다고 하는데, 무척 기대된다. 그러나 요즘 연구들이 모델 자체의 특출한 아이디어보다는 분산처리나 대규모 데이터 처리에 관한 기술 발전에 기반해서 더 크고 많이 학습시킨 모델일 뿐 모델 구조적이나 어떤 특수한 패러다임이 등장하지 않는 약간은 정체한다는 느낌이 들기도 한다.