자연어처리

컴퓨터공학과 19011547 김상원

[주제: 우울증 진단]

[제출일: 2023-06-25]

목차

1. 주제 선정

어렸을 때는 정신병의 원인은 나약한 마음가짐 때문이고, 이를 치료하는 의사는 소위 장사꾼이라고 생각했습니다. 하지만 고등학생 때부터 사귄 전 여자친구가 공황장애와 우울증에 걸렸고 함께 정신병원을 다니며 저의 인식이 바뀌었습니다. 그 친구는 정신병원을 다닌 이력이 남을 것이 두려워, 상태가 악화된 후에 병원에 갔고 고생을 많이 했습니다. 그래서 만약 그 친구의 병을 보다 빨리 비대면으로 진단받았더라면 고생을 덜했을 거라는 안타까움이 있습니다.

현재 대중적으로 알려진 우울증 간이 진단지(PHQ-9)는 간이 설문조사이기 때문에, 맥락을 파악하기 힘들다는 한계점이 있습니다. 또한 병원에서 진행하는 상담은 보통 해당 간이진단지를 바탕으로 진행되는 데, 한 명의 의사가 많은 환자를 진찰해야 하기 때문에 환자에 대한 정보를 최대한 빨리 파악해야 보다 깊이 있는 진단을 할 수 있습니다.

따라서 저는 위의 문제를 해결하고자 합니다. 자신의 감정을 표현할 수 있게 서술형 답변을 하고, 의사들은 환자의 정보를 쉽게 파악할 수 있도록 이를 요약할 수 있어야 합니다. 그래서 서술형 답변의 감정을 분류할 수 있는 LSTM 모델과 해당 답변을 기반으로 의사들이 필요한 정보만 추출할 수 있는 distilbert 기반 모델을 만들었습니다.

2. 훈련 데이터과 LSTM을 활용한 우울증 진단

훈련 데이터

AIHub의 감성 말뭉치[1]를 활용하였습니다. 질의응답형식의 데이터는 화자와 청자의 관계에서 주로 화자의 응답을 유도하는 식의 질문으로 통해 답변을 이끌어냅니다. 따라서 해당 데이터에서 화자의 감정 문장만 활용합니다. 이때 화자의 감정은 대분류 6가지, 소분류 60가지의 감정으로 구성됩니다. 따라서 '분노', '슬픔', '불안', '상처'를 '우울한 감정'을 의미하는 1로 '기쁨', '당황'을 '우울하지 않은 감정'을 의미하는 0으로 재분류했습니다. 대분류 6개의 데이터를 4:2의 비율로 다시 재분류를 했기 때문에 후에 모델 훈련을 할 때, 가중치를 다르게 줘서 overfitting을 방지하였습니다

[1] 과학기술정보통신부, AIhub, 감성 말뭉치 문자데이터 https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=86

LSTM

심리학에서는 맥락이 중요하고, 제가 구한 데이터도 화자가 자신의 감정 변화를 3번의 응답으로 대답하기 때문에 LSTM을 적용하기 좋은 데이터라고 판단하였습니다. 또한 훈련 데이터의 경우 긴 문장으로 구성되어 있고, 우울증 진단에 필요한 응답도 긴 문장으로 들어올 것을 기대하기 때문에 단순한 RNN 모델로는 감정을 예측하기 어렵다고 판단되었습니다. 전체 입력 데이터를 대상으로 time_step로 훈련함으로써, 기존 검사지의 한계점인 맥락을 파악할 수 없다는 단점을 극복하여 심리 분석에 신뢰성을 높였습니다.

.

3. LSTM 코드 구현

데이터 전처리

MAX_LEN을 확인한다.

```
lengths = [len(d) for d in tmp]
sorted_lengths = sorted(lengths)
```

```
np.mean(sorted_lengths)
```

```
36.30440312479824
```

```
[28] MAX_LEN = round(np.mean(sorted_lengths)) # 표를 봤을 때, 평균이 가장 합리적이다.
```

```
def morphizing_corpos(train, tokenize_as_morph = False):
  if tokenize_as_morph: # tokenize
    morphized_corpos = prepro_like_morphized(train['corpos'])
    corpos_len_by_morph = [len(t) for t in morphized_corpos]
    corpos_len_by_morph_mean = int(np.mean(corpos_len_by_morph)) # 평균 문장 길이
    print("corpos_len_by_morph_mean: ", corpos_len_by_morph_mean)
  return (morphized_corpos, corpos_len_by_morph_mean)
```

LSTM은 정해진 길이의 문장만 입력 받을 수 있습니다. 따라서 해당 데이터의 평균 길이와 분포를 봤을 때, 평균 길이로 하는 것이 가장 합리적이라고 판단하여 평균 문장 길이를 입력의 단위로 정했습니다. 이외에 자연어 토큰화하는 과정은 Okt()으로 한글을 분리하고 학습에 불필요한 데이터를 삭제하는 등의 일반적인 방식으로 진행하였습니다.

사전 생성과 padding

```
[46] def load_vocabulary(path, train): # path = '/content/drive/MyDrive/자연어처리 과제/vocabulary.txt', train = trainX
       morphized_corpos = []
       (morphized_corpos, MAX_LEN) = morphizing_corpos(train, True)
       vocabulary = data_tokenizer(morphized_corpos)
       vocabulary[:0] = MARKER
       with open(path, 'w', encoding='utf-8') as vocabulary_file:
         for word in vocabulary:
           vocabulary_file.write(word + '\n')
       vocabulary_list = []
       with open(path, 'r', encoding='utf-8') as vocabulary_file:
         for line in vocabulary_file:
           vocabulary_list.append(line.strip())
       vocabulary = make_vocabulary(vocabulary_list)
       return (vocabulary, MAX_LEN)
```

```
def pad_sequence(train, dictionary, MAX_LEN = 113, tokenize_as_morph = False):
  sequence_input_index = []

  if tokenize_as_morph:
    train = prepro_like_morphized(train)
  for sequence in train:
    sequence = re.sub("([~.,!?\"':;)(])", "", sequence)
    sequence_index = []
    for word in sequence.split():
      if dictionary.get(word) is not None:
        sequence_index.extend([dictionary[word]])
      else:
        sequence_index.extend([dictionary[UNK]])
    if len(sequence_index) > MAX_LEN:
      sequence_index = sequence_index[:MAX_LEN]
    else:
      sequence_index += (MAX_LEN - len(sequence_index)) * [dictionary[PAD]]
    sequence_input_index.append(sequence_index)
  return np.array(sequence_input_index)
```

colab의 자원을 아끼기 위하여 사전을 생성하면 이후부터는 불러오는(load)하는 방식으로 진행하였습니다. Train 데이터로 생성한 사전을 활용하여 one-hot encoding을 진행하였습니다.

LSTM으로 학습시키기

입력 데이터 3개를 하나로 합친 후에 LSTM으로 훈련을 시켰습니다. 기존 LSTM보다 양방향 LSTM의 성능이 더 좋아서 Bidirectional(LSTM)을 2개 사용하였습니다. 또한 데이터 비율이 다르기 때문에 overfiting을 방지하고자 가중치를 조절하였습니다.

class의 비중이 맞지 않기 때문에 weight에 가중치를 준다.

```
[ ]  class_weights = {}
     class_weights[0] = len(trainY) / (2 * np.sum(trainY == 0))
     class_weights[1] = len(trainY) / (2 * np.sum(trainY == 1))
     trainY = trainY.astype(np.float32)
     testY = testY.astype(np.float32)
     validY = validY.astype(np.float32)
     print(trainX_padded.shape)
     print(class_weights[0], class_weights[1], np.sum(trainY == 0), np.sum(trainY == 1))
```

```
model = Sequential([
    Embedding(vocab_size, embedding_dim, input_length = MAX_LEN),
    Bidirectional(LSTM(128, return_sequences = True)),
    Bidirectional(LSTM(128)),
    Dropout(0.25),
    Dense(32, activation = 'relu'),
    Dense(1, activation = 'sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics = ['acc'])
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(testY, prediction))
```

```
0.8182502635145309
```

4. 훈련 데이터와 BERT을 활용한 기계 독해

훈련 데이터

KorQuAD의 질문응답쌍 데이터를 활용하겠습니다. 해당 데이터는 Wikipedia article을 기반으로 해당 article에 관한 질문과 답변쌍으로 구성되어 있습니다. 해당 데이터는 기계 독해을 위한 모델 개발을 위해 사용됩니다. 해당 데이터의 양이 방대하고 기계 독해 모델을 만들기 위해 사용할 pretraining model인 bert의 용량이 커서 colab으로 훈련시키기에 어려움이 많았습니다. 따라서 훈련 데이터로 인한 RAM 사용량을 줄이기 위해 preprocessing을 사전에 하고 preprocessing이 종료된 데이터는 numpy로 저장했습니다. 그리고 RAM을 초기화하고 이전에 저장했던 numpy를 불러와서 학습에 활용하였습니다.

distilbert-base-multilingual-cased

초기에는 교안에 있는 bert-base-multilingual-cased를 활용하고자 하였으나 Colab의 GPU와 System RAM이 작은 관계로 많은 시도를 하였으나 학습 자체가 불가능하였습니다. 따라서 보다 경량화 모델인 distilbert-base-multilingual-cased을 전이학습 모델로 사용하기로 결정하였습니다. 또한 경량화 모델로 학습을 진행시켜도 계속 실패하여, 모델의 학습 데이터를 줄이는 대신에 Sklearn의 VotingClassifier모델과 유사한 앙상블 모델을 만들기로 결정하였습니다.

5. Pretrained-Bert 코드 구현

colab으로 코드 구현의 어려운점

```
    File "/usr/local/lib/python3.10/dist-packages/keras/backend.py", line 2171, in dropout
      return tf.nn.dropout(
Node: 'tf_distil_bert_question_answering/tf_distil_bert_model/distilbert/transformer/layer_._5/attention/dropout_17/dropout/random_uniform/RandomUniform'
OOM when allocating tensor with shape[47047,12,384,384] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_bfc
	 [[{{node tf_distil_bert_question_answering/tf_distil_bert_model/distilbert/transformer/layer_._5/attention/dropout_17/dropout/random_uniform/RandomUniform}}]]
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom to RunOptions for current allocation info. This isn't available when running in Eager mode.
 [Op:__inference_train_function_15042]
```

SEARCH STACK OVERFLOW

distilbert-base-multilingual-cased를 도입하다.

초기에는 교안에 있는 bert-base-multilingual-cased를 구현하여, 기계 독해를 할 수 있는 인공지능 모델을 프로젝트에 넣게 되면 완성도가 높아질 것이라고 생각했습니다. 하지만 해당 모델을 그대로 구현하면 colab의 메모리 부족 문제가 발생했습니다. 따라서 distilbert-base-multilingual-cased를 보다 경량화 버전인 도입하여 교안 내용을 최대한 참고하며 구현하였습니다.

앙상블을 도입하다.

distilbert-base-multilingual-cased를 도입하였으나 계속 메모리 부족현상이 발생하였습니다. 따라서 한번에 모델이 학습하는 데이터 양도 줄이고자 하였습니다. 따라서 데이터를 5분할하여 그 중 2개의 데이터를 사용하고, 모델을 2개를 만들어 각각의 모델이 적은 데이터로 학습하여 GPU의 가부화는 줄이지만 성능을 최대한 보존하고자 하였습니다.

파악할 수 없는 에러가 발생하다.

위와 같은 방식으로 학습을 완료할 수 있었지만, 교안을 참고하여 코드를 작성하였으나 교안과 다른 모델을 pretraining 모델로 사용했기 때문에 에러가 계속 발생했습니다. 며칠 간 계속 고쳤지만, 입력부분에서 에러가 발생하지 않으면 예측 값이 이상했습니다.

```
model: 0
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_1/bert/pooler/dense/kernel:0', 'tf_bert_model_1/bert/pooler/dense/bias:0'] when minimizing the loss. If you're using `model.compile()`, did you forget to provide a `loss` argu
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_1/bert/pooler/dense/kernel:0', 'tf_bert_model_1/bert/pooler/dense/bias:0'] when minimizing the loss. If you're using `model.compile()`, did you forget to provide a `loss` argu
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_1/bert/pooler/dense/kernel:0', 'tf_bert_model_1/bert/pooler/dense/bias:0'] when minimizing the loss. If you're using `model.compile()`, did you forget to provide a `loss` argu
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_1/bert/pooler/dense/kernel:0', 'tf_bert_model_1/bert/pooler/dense/bias:0'] when minimizing the loss. If you're using `model.compile()`, did you forget to provide a `loss` argu
138/138 [==============================] - 128s 903ms/step
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
```

결국 완성했습니다.

```
print(ans)

[['1989년 2월 15일', '1989년 2월 15일 여의도 농민 폭력 시위를 주도한 혐의(폭력행위등처벌에관한법률위반)으로 지명수배되었다. 1989년 3월 12일 서울지방
```

```
def create_inputs_targets(squad_examples):
    dataset_dict = {
        "input_ids": [],
        "attention_mask": [],
        "start_token_idx": [],
        "end_token_idx": [],
    }
    for item in squad_examples:
        if item.skip == False:
            for key in dataset_dict:
                dataset_dict[key].append(getattr(item, key))
    for key in dataset_dict:
        dataset_dict[key] = np.array(dataset_dict[key])

    x = [
        dataset_dict["input_ids"],
        dataset_dict["attention_mask"],
    ]
    y = [dataset_dict["start_token_idx"], dataset_dict["end_token_idx"]]
    return x, y
```

해당 코드를 작성하는 중에 생긴 에러 중 찾는데 가장 오래 걸렸던 부분은 'token_type_ids'입니다. 왜냐하면 distilbert-base-multilingual-cased에서는 token_type_ids를 사용하지 않기 때문입니다.

```
for i in range(0, 2):
  x_train_subset = x_train[:, fold * i: fold * (i + 1), :]
  y_train_subset = y_train[:, fold * i: fold * (i + 1)]
  optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
  model_array[i].compile(optimizer=optimizer, loss=[loss, loss])
  print("model:",i)
  history = model_array[i].fit(
      # x = x_train_subset.tolist(),
      # y = y_train_subset.tolist(),
      x = [x_train_subset[0, :, :], x_train_subset[2, :, :]],
      y = [y_train_subset[0, :], y_train_subset[1, :]],
      epochs=EPOCHS,
      batch_size=BATCH_SIZE,
      callbacks=[exact_match_callback, cp_callback] # 모델 평가 및 가중치 저장
  )
  history_list.append(history)
```

해당 부분은 앙상블을 구현하기 위해 구현한 부분입니다. 비록 앙상블의 성능이 예상과 달리 좋지 못하였습니다. 최선을 다했습니다.

```
[ ]  pred_start, pred_end = model_array[0].predict([x_eval_[0, :, :], x_eval_[1, :, :]])
     # eval_examples_no_skip = [_ for _ in eval_squad_examples if _.skip == False]
     for idx, (start, end) in enumerate(zip(pred_start, pred_end)):
         # squad_eg = eval_examples_no_skip[idx]
         # offsets = squad_eg.context_token_to_char
         start = np.argmax(start)
         end = np.argmax(end)
         first_model.append((start, end))

     138/138 [==============================] - 61s 443ms/step
```

```
[ ]  pred_start, pred_end = model_array[1].predict([x_eval_[0, :, :], x_eval_[1, :, :]])
     eval_examples_no_skip = [_ for _ in eval_squad_examples if _.skip == False]
     for idx, (start, end) in enumerate(zip(pred_start, pred_end)):
         squad_eg = eval_examples_no_skip[idx]
         offsets = squad_eg.context_token_to_char
         start = np.argmax(start)
         end = np.argmax(end)
         second_model.append((start, end))

     138/138 [==============================] - 62s 447ms/step
```

```
     eval_examples_no_skip = [_ for _ in eval_squad_examples if _.skip == False]
     idx = 0
     count = 0
     for i in range(len(first_model)):
       squad_eg = eval_examples_no_skip[idx]
       offsets = squad_eg.context_token_to_char
       idx+=1
       s_1, e_1 = first_model[i]
       s_2, e_2 = second_model[i]
       start = round((s_1 + s_2)/2)
       end = round((e_1 + e_2)/2)
       if start >= len(offsets):
         continue
       pred_char_start = offsets[start][0]
       if end < len(offsets):
           pred_char_end = offsets[end][1]
           pred_ans = squad_eg.context[pred_char_start:pred_char_end]
       else:
           pred_ans = squad_eg.context[pred_char_start:]
       normalized_pred_ans = normalized_answer(pred_ans)
       normalized_true_ans = normalized_answer(squad_eg.answer_text)
       if normalized_pred_ans in normalized_true_ans:
           count += 1
     acc = count / len(y_eval[0])
     print(f"exact match score={acc:.2f}")

     exact match score=0.57
```

이후에 앙상블을 통한 결과를 에측한 결과입니다. 앙상블의 결과가 오히려 떨어졌기 때문에 하나의 모델만 사용하기로 하였습니다.

flask에 사용하는 것에 실패하다.

```
reprocessing.py
Traceback (most recent call last):
  File "c:\Users\82105\Desktop\자연어처리\bert_preprocessing.py", line 35, in <module>
    tokenizer = BertWordPieceTokenizer(save_path + "vocab.txt", lowercase=False)
Downloading (…)solve/main/vocab.txt: 232kB [00:00, 623kB/s]
C:\ProgramData\anaconda3\envs\myenv\lib\site-packages\huggingface_hub\file_download.py:133: UserWarning: `huggingface_hub` cache-s
ystem uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\82105\.cac
he\huggingface\hub. Caching files will still work but in a degraded version that might require more space on your disk. This warni
ng can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingfac
e.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to see a
ctivate developer mode
```

에러를 잡고, 해당 모델을 local에서 돌리기 위해 작업해야 할 부분이 많아 시간이 많이 흘렀습니다. 하지만 위 그림과 같은 에러가 발생했고 구글링을 해본 결과, 제 노트북에서는 해당 버전의 파일을 허용하지 않는다고 합니다. 물론 고칠 수는 있겠으나, 시간 관계상 포기할 수 밖에 없었습니다.

6. Flask를 활용한 구현

```python
@app.route('/questionnaire_answer', methods=['GET', 'POST'])
def ft_questionnaire_answer():
    context = ""
    answer = {'frequency':[], 'message':[]}
    if request.method == 'POST':
        for i in range(1, 5):
            # frequency = int(request.form['checker' + str(i)]) # 감정에 대한 답변을 받기 때문에 필요 없다.
            frequency = 4
            if i == 1:
                frequency = frequency * 2 # 문항 2개를 합쳤기 때문에 가중치를 준다.
            message = request.form['message' + str(i)]
            answer['frequency'].append(frequency)
            answer['message'].append(message)
            context = context + " " + message
        print(answer)
        collected_data['answer'] = answer
        if len(answer) == 4:
            print("Data submitted succesffully", answer)

        # user_submit['data'][0]['paragraphs'][0]['context'] = context
        return redirect('/questionnaire_check')
    return render_template('questionnaire_answer.html')
```

```python
@app.route('/questionnaire_check', methods=['GET', 'POST'])
def ft_questionnaire_check():
    check = []
    if request.method == 'POST':
        for i in range(1, 5):
            frequency = int(request.form['checker' + str(i)])
            check.append(frequency)
        # Process the data as needed (e.g., append to a CSV file)
        collected_data['check'] = check
        # Process the data as needed (e.g., append to a CSV file)
        if len(check) == 4:
            print("Data submitted succesffully", check)
        else:
            return "Data submitted erorr in questionnaire_check"
        return redirect('/result')
    return render_template('questionnaire_check.html')
```

설문지는 총 2종류로 구성됩니다. 감정에 대해 서술할 수 있는 주관식과 감정이 아닌 빈도에 대한 답변을 해야 하는 객관식으로 구성됩니다. 우선 서술형 답변은 자연어처리를 통해 감정 분류를 통해 점수를 매기고 객관식 답변을 통해 얻은 점수를 더하여 우울증 점수를 산출합니다. 해당 결과는 result.html로 전달합니다.

7. 한계점

우울증에 대해 관심은 있었으나 지식이 부족하여 심리학 석사 과정을 밟고 있는 친구와 학부 전공을 하고 있는 두 명의 친구에게 많이 물어봤고, 도움을 많이 받았습니다. 우울증 검사에서 중요한 것은 문장 단위의 의미 파악 보다는 맥락이 중요하다는 애기를 공통적으로 들었고, 어떻게 하면 맥락을 잘 파악할 수 있을지 고민을 많이 하였습니다. 이를 통해 대략적인 기준을 갖고 모델 구현을 할 수 있었으나 완전하지는 않습니다.

기계 독해의 한계점은 colab을 사용한다는 한계점으로 인해, 학습을 시키는 것보다 학습을 분산시키는 것에 지나치게 시간을 많이 썼습니다. 또한 사전 학습 모델을 처음 사용하여 local에서 설정하는 것에 너무 많이 시간이 걸렸습니다. 또한 에러를 처리하는 것도 어려워 마감기한까지 모델 구현은 완료하였으나 flask에 결합하지 못하여 매우 아쉽습니다.