한국어로 이루어진 문장 기반의 다중 감정 분석

윤예준

INDEX

01 소개

02 데이터
전처리

03 실험

04 결과

01

소개

- 동기

- 데이터셋 – AI HUB 감성 대화 말뭉치

	A	B	C	D	E	F	G	H	I	J
1	번호	연령	성별	상황키워드	신체질환	감정_대분류	감정_소분	사람문장1	시스템응답1	사람문장2
2	44164	청년	남성	연애, 결혼, 출산	해당없음	기쁨	신이 난	아내가 드디어 출산	아내분이 출산을 하	아 지금 정말 신이
3	3926	노년	남성	건강, 죽음	만성질환 -	불안	스트레스	당뇨랑 합병증 때문	약 종류가 많아 번	건강할 때 관리 좀
4	50882	청소년	여성	학업 및 진로	해당없음	당황	당황	고등학교에 올라오	고등학교 수업이 좋	아직 학기 초인데
5	31303	노년	남성	재정	만성질환 -	기쁨	신이 난	재취업이 돼서 받	재취업 후 첫 월급	퇴직 후 다시는 돈

그림
1.

- 감정_대분류: 기쁨, 당황, 분노, 불안, 상처, 슬픔

02

데이터 전처리

```
df_train = df_train.iloc[:40879,:15]

df_train.replace(np.nan, '', inplace=True)

ls = {}
df = pd.DataFrame(ls)
df['문장'] = df_train[['사람문장1', '시스템응답1', '사람문장2','시스템응답2',
                       '사람문장3','시스템응답3','사람문장4','시스템응답4']].apply(' '.join, axis =1)

df['감정'] = df_train['감정_대분류']
```

그림 2.

	문장	감정
0	아내가 드디어 출산하게 되어서 정말 신이 나. 아내분이 출산을 하시는군요. 정말 축...	기쁨
1	당뇨랑 합병증 때문에 먹어야 할 약이 열 가지가 넘어가니까 스트레스야. 약 종류가 ...	불안
2	고등학교에 올라오니 중학교 때보다 수업이 갑자기 어려워져서 당황스러워. 고등학교 수...	당황
3	재취업이 돼서 받게 된 첫 월급으로 온 가족이 외식을 할 예정이야. 너무 행복해. ...	기쁨
4	빚을 드디어 다 갚게 되어서 이제야 안도감이 들어. 기분 좋으시겠어요. 앞으로는 어...	기쁨
...	...	...

그림 3.

문장: 사람문장1, 시스템응답1, 사람문장2, 시스템응답2…시스템응답4

```
1 le = LabelEncoder()
2 le.fit(df['감정'])
3 print(le.classes_)
4 df['감정_label']=le.transform(df['감정'])
```

그림 4.

```
1 df['감정_label'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc65d90bcd0>
```

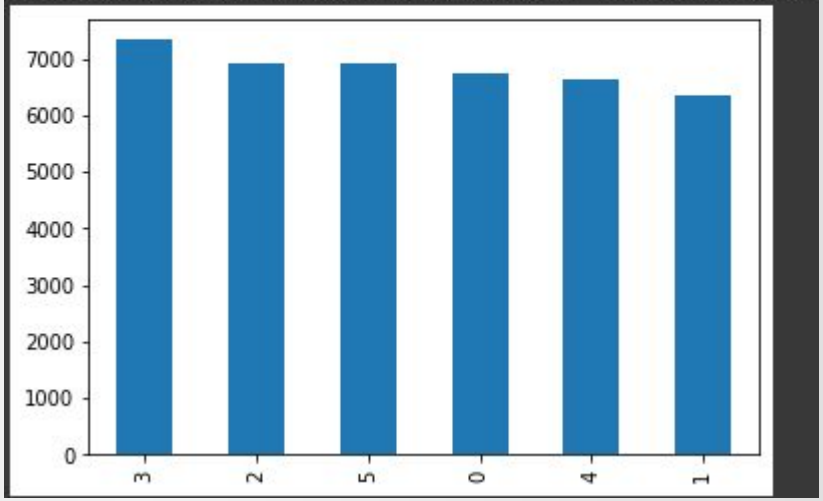

그림 5.

```
df['문장'].replace('', np.nan, inplace=True)
print(df.isnull().values.any())
print(df.isnull().sum())
df = df.dropna(how = 'any')
```

```
False
문장    0
감정    0
dtype: int64
```

```
# 한글과 공백을 제외하고 모두 제거
df['문장'] = df['문장'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
```

그림 6.

```
train_data, test_data = train_test_split(df, test_size = 0.2, random_state = 11)
```

```
stopwords = ['도', '는', '다', '의', '가', '이', '은', '한',
             '에', '하', '고', '을', '를', '인', '듯', '과',
             '와', '네', '들', '듯', '지', '임', '게', '것',
             '어', '겠', '있', '군요', '해']

mecab = Mecab()

train_data['tokenized'] = train_data['문장'].apply(mecab.morphs)
train_data['tokenized'] = train_data['tokenized'].apply(lambda x: [item for item in x if item not in stopwords])
test_data['tokenized'] = test_data['문장'].apply(mecab.morphs)
test_data['tokenized'] = test_data['tokenized'].apply(lambda x: [item for item in x if item not in stopwords])
```

그림 7.

```
# np.array로 반환
X_train = train_data['tokenized'].values
y_train = train_data['감정_label'].values
X_test= test_data['tokenized'].values
y_test = test_data['감정_label'].values
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
print(tokenizer.word_index)
```

```
{'시': 1, '좋': 2, 'ㄴ': 3, '친구': 4, '아': 5, '어요
```

```
tokenizer = Tokenizer(len(tokenizer.word_index))
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

그림 8.

```
def below_threshold_len(max_len, nested_list):
  cnt = 0
  for s in nested_list:
    if(len(s) <= max_len):
        cnt = cnt + 1
  print('전체 샘플 중 길이가 %s 이하인 샘플의 비율: %s'%(max_len, (cnt / len(nested_list))*100))
```

```
max_len = 110
below_threshold_len(max_len, X_train)
```

```
전체 샘플 중 길이가 110 이하인 샘플의 비율: 99.9021496498792
```

```
X_train = pad_sequences(X_train, maxlen = max_len)
X_test = pad_sequences(X_test, maxlen = max_len)
```

그림 10.

```
print('문장의 최대 길이 :',max(len(l) for l in X_train))
print('문장의 평균 길이 :',sum(map(len, X_train))/len(X_train))
plt.hist([len(s) for s in X_train], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

```
문장의 최대 길이 : 141
문장의 평균 길이 : 55.43968443262086
```

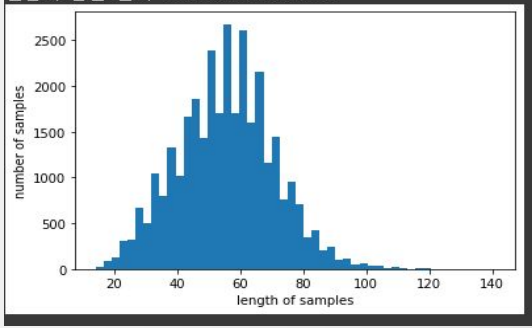

그림 9.

```
1 from gensim.models import Word2Vec
2 Word2Vec_model = Word2Vec(sentences = train_data['tokenized'], size = 300, window = 5,
3                           min_count = 5, workers = 4, sg = 0)
4 Word2Vec_model.wv.vectors.shape
```

```
(8600, 300)
```

```
1 vocab_size = len(tokenizer.word_index)+1
2 embedding_matrix = np.zeros((vocab_size, 300))
3 print(np.shape(embedding_matrix))
```

그림 11.

```
1 def get_vector(word):
2     if word in Word2Vec_model:
3         return Word2Vec_model[word]
4     else:
5         return None
6
7 for word, i in tokenizer.word_index.items(): # 훈련 데이터의 단어 집합에서 단어와 정수 인덱스를 1개씩 꺼내온다.
8     temp = get_vector(word) # 단어(key) 해당되는 임베딩 벡터의 300개의 값(value)를 임시 변수에 저장
9     if temp is not None: # 만약 None이 아니라면 임베딩 벡터의 값을 리턴받은 것이므로
10        embedding_matrix[i] = temp # 해당 단어 위치의 행에 벡터의 값을 저장한다.
```

그림 12.

03

실험

Dense층만 사용

```
model = models.Sequential()
e = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable=False)
model.add(e)
model.add(Flatten())
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(6, activation ='softmax'))
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])

history = model.fit(X_train, one_hot_train_labels , epochs=5, batch_size=128
                    , validation_data=(X_test, one_hot_test_labels))
```

```
Epoch 3/5
256/256 [==============================] - 2s 7ms/step - loss: 0.6932 - acc: 0.7473 - val_loss: 1.6877 - val_acc: 0.4576
Epoch 4/5
256/256 [==============================] - 2s 7ms/step - loss: 0.4195 - acc: 0.8527 - val_loss: 2.0562 - val_acc: 0.4463
Epoch 5/5
```

LSTM 사용

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras.layers import Flatten, Embedding, LSTM


model = models.Sequential()
e = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable=False)
model.add(e)
model.add(LSTM(128))
model.add(Flatten())
# model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(6, activation ='softmax'))
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])

history = model.fit(X_train, one_hot_train_labels , epochs=5, batch_size=128
                    , validation_data=(X_test, one_hot_test_labels))
```

```
Epoch 1/5
256/256 [==============================] - 9s 17ms/step - loss: 1.3459 - acc: 0.4691 - val_loss: 1.1398 - val_acc: 0.5851
Epoch 2/5
256/256 [==============================] - 4s 15ms/step - loss: 0.9806 - acc: 0.6529 - val_loss: 0.9440 - val_acc: 0.6712
Epoch 3/5
256/256 [==============================] - 4s 15ms/step - loss: 0.8691 - acc: 0.6967 - val_loss: 0.8865 - val_acc: 0.6931
Epoch 4/5
256/256 [==============================] - 4s 15ms/step - loss: 0.8050 - acc: 0.7188 - val_loss: 0.8785 - val_acc: 0.6947
Epoch 5/5
256/256 [==============================] - 4s 15ms/step - loss: 0.7577 - acc: 0.7369 - val_loss: 0.8860 - val_acc: 0.6953
```

CNN 사용

```python
from tensorflow.keras import models, Model
from tensorflow.keras import layers
from tensorflow.keras.layers import Flatten, Embedding, LSTM, Input, Dropout, Conv1D, GlobalMaxPooling1D, Concatenate, Dense

dropout_prob = (0.5, 0.8)
num_filters = 128

model_input = Input(shape = (max_len,))
z = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable=False)(model_input)
z = Dropout(dropout_prob[0])(z)

conv_blocks = []

for sz in [3, 4, 5]:
    conv = Conv1D(filters = num_filters,
                          kernel_size = sz,
                          padding = "valid",
                          activation = "relu",
                          strides = 1)(z)
    conv = GlobalMaxPooling1D()(conv)
    conv = Flatten()(conv)
    conv_blocks.append(conv)

z = Concatenate()(conv_blocks) if len(conv_blocks) > 1 else conv_blocks[0]
z = Dropout(dropout_prob[1])(z)
z = Dense(128, activation="relu")(z)
model_output = Dense(6, activation="sigmoid")(z)

model = Model(model_input, model_output)
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])


model.fit(X_train, one_hot_train_labels, batch_size = 64, epochs=10, validation_data = (X_test,  one_hot_test_labels), verbose=2)
```

```
Epoch 7/10
511/511 - 4s - loss: 1.2405 - acc: 0.5230 - val_loss: 1.1026 - val_acc: 0.5818
Epoch 8/10
511/511 - 4s - loss: 1.2126 - acc: 0.5430 - val_loss: 1.0723 - val_acc: 0.6148
Epoch 9/10
511/511 - 4s - loss: 1.1839 - acc: 0.5592 - val_loss: 1.0463 - val_acc: 0.6296
Epoch 10/10
511/511 - 4s - loss: 1.1749 - acc: 0.5674 - val_loss: 1.0381 - val_acc: 0.6333
```

BILSTM 사용

```
model = models.Sequential()
e = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=max_len, trainable=False)
model.add(e)
model.add(Bidirectional(LSTM(100)))
model.add(layers.Dense(6, activation ='softmax'))

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])

model.fit(X_train, one_hot_train_labels, batch_size = 64, epochs=5, validation_data = (X_test,  one_hot_test_labels))
```

```
Epoch 4/10
511/511 [==============================] - 9s 17ms/step - loss: 0.7781 - acc: 0.7307 - val_loss: 0.8590 - val_acc: 0.6985
Epoch 5/10
511/511 [==============================] - 8s 16ms/step - loss: 0.7340 - acc: 0.7461 - val_loss: 0.8484 - val_acc: 0.7100
Epoch 6/10
511/511 [==============================] - 9s 17ms/step - loss: 0.6942 - acc: 0.7604 - val_loss: 0.8551 - val_acc: 0.7095
Epoch 7/10
511/511 [==============================] - 9s 17ms/step - loss: 0.6549 - acc: 0.7762 - val_loss: 0.8638 - val_acc: 0.7040
Epoch 8/10
511/511 [==============================] - 9s 18ms/step - loss: 0.6121 - acc: 0.7915 - val_loss: 0.8705 - val_acc: 0.7067
Epoch 9/10
511/511 [==============================] - 9s 18ms/step - loss: 0.5672 - acc: 0.8074 - val_loss: 0.9170 - val_acc: 0.6958
```

04

결론 및 한계점

04

결론

THE END

감사합니다