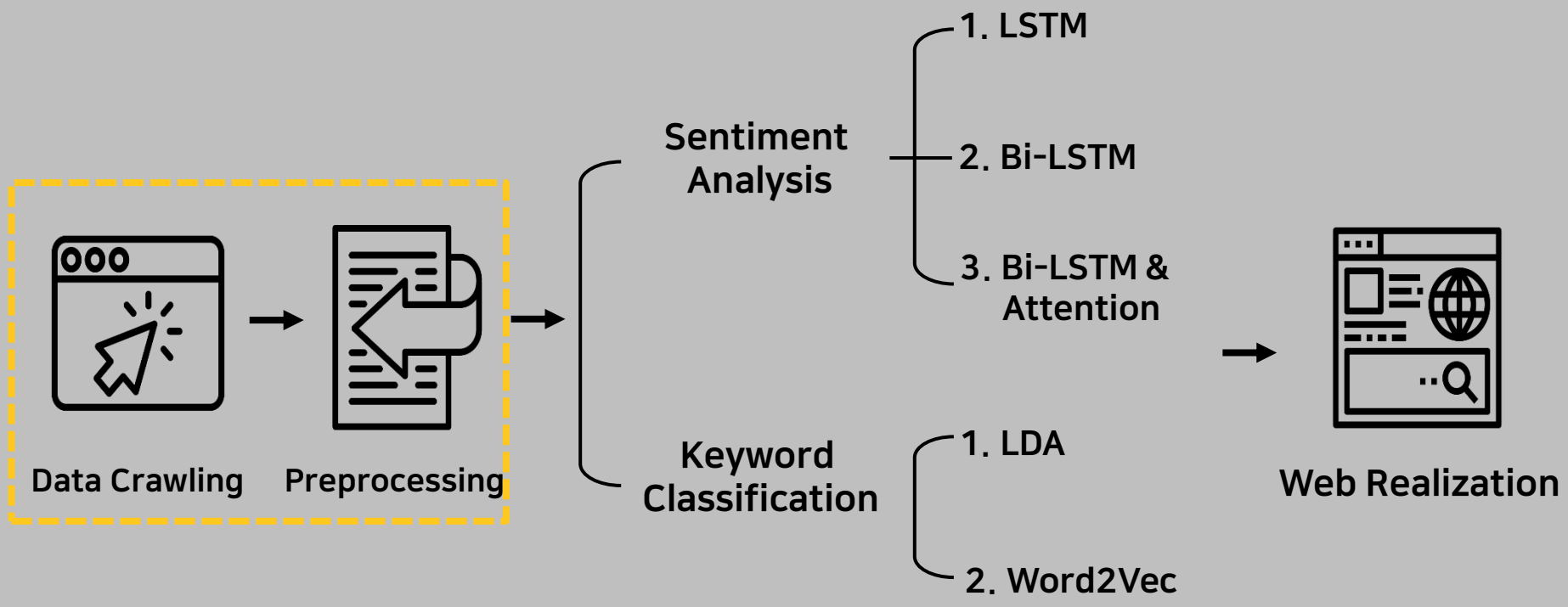
감성 분석, 키워드 분류 기반
카페 평가 및 추천 프로젝트

Ybigta 18기 김승하, 한지영, 홍진우

목차(Index)

01 프로젝트 개요 및 목표



01 프로젝트 개요 및 목표

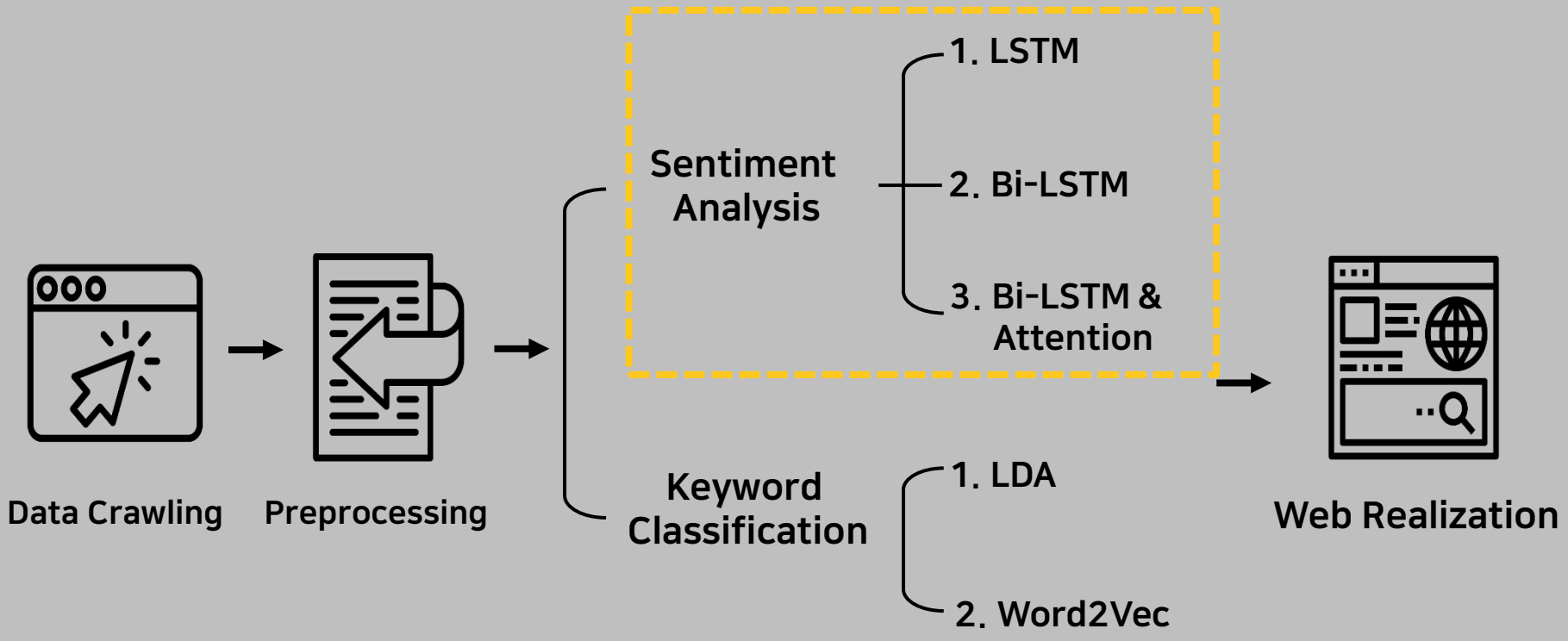
- 3만 5천개의 서울소재 카페 리뷰를 바탕으로 **감성 분석**, **키워드 분류 모델** 구현

	row_num	location	cafe_name	review	label
0	0	강남역	카페노티드 청담	그냥 그렇다 생각했는데 다른 곳에서 도넛 먹고나면? ㄷ ㄷ 킹티드 .. 잘되는 이유...	4
1	1	강남역	카페노티드 청담	우유크림 개맛있슴 근데 그만큼 혼잡해서 또 가고싶지 않은 곳..	3
2	2	강남역	카페노티드 청담	도넛은 맛있는데 종업원들 대응이 너무 느려요. 한남이랑 삼성점도 가봤는데 여기가 유...	2
3	3	강남역	카페노티드 청담	도넛이 맛은 있는데 줄이 진짜 너엉어ㅓㅓ어어어무 길다 예약해서 가져갈거면 가고 가...	4
4	4	강남역	카페노티드 청담	감사합니다	1
5	5	강남역	카페노티드 청담	바닐라맛 맛있어요! 도넛 사는데.. 40분? 기다린거 같네요 . 이왕 간김에 조각 ...	4
6	6	강남역	카페노티드 청담	도넛 맛은 맛있음 대신 포장해서 갈거면 인내심 길러가야함 뭔가 포장하는 방식이 바뀌...	3
7	7	강남역	카페노티드 청담	계산해주는 직원도 불친절하고 크로플도 두개시켰는데 집에 와서 열어보니 한개만 줬네요...	1

- **유저 친화적**인 정보 제공 및 시각화(별점이 누락된 리뷰 라벨링, 키워드별 리뷰 확인)

- 유저가 남긴 리뷰에 대한 감성분석, 키워드 분석을 통해 **맞춤형 카페 추천 알고리즘** 구현

02 감성분석 모델


Data Crawling
Preprocessing
Sentiment Analysis
1. LSTM
2. Bi-LSTM
3. Bi-LSTM & Attention
Keyword Classification
1. LDA
2. Word2Vec
Web Realization

01 데이터 셋

	row_num	location	cafe_name	review	LSTM_pp	label_org	label_mod
0	0	강남역	카페노티드 청담	그냥 그렇다 생각했는데 다른 곳에서 도넛 먹고나면? ㄷ ㄷ 킹티드 .. 잘되는 이유...	그냥/Noun,그렇다/Adjective,생각/Noun,하다/Verb,다른/Noun,...	4	1
1	1	강남역	카페노티드 청담	우유크림 개맛있슴 근데 그만큼 혼잡해서 또 가고싶지 않은 곳..	우유/Noun,크림/Noun,개/Noun,맛있다/Adjective,근데/Adverb...	3	3
2	2	강남역	카페노티드 청담	도넛은 맛있는데 종업원들 대응이 너무 느려요. 한남이랑 삼성점도 가봤는데 여기가 유...	도넛/Noun,맛있다/Adjective,종업원/Noun,대응/Noun,너무/Adve...	2	0
3	3	강남역	카페노티드 청담	도넛이 맛은 있는데 줄이 진짜 너엉어ㅓㅓ어어어어무 길다 예약해서 가져갈거면 가고 가...	도넛/Noun,맛/Noun,있다/Adjective,줄이다/Verb,진짜/Noun,너...	4	1
4	4	강남역	카페노티드 청담	감사합니다	감사하다/Verb	1	0
...	...	...	...	...	...	...	...
34484	34484	한남동	모모	조금 아쉬워요	조금/Noun,아쉽다/Adjective	2	0
34485	34485	한남동	모모	우리는 결국 모두 어떤 기대 없이 하도록 돼 있는 일을 하게 된다. 그래서 이 곳은...	우리/Noun,결국/Adverb,모두/Noun,어떻다/Adjective,기대/Nou...	5	1
34486	34486	한남동	모모	가격도 합리적이고 분위기두 굿!	가격/Noun,합리/Noun,분위기/Noun,후/Noun,굿/Noun	5	1
34487	34487	한남동	모모	조용한 골목에 있어서 일단 너무 좋아요. 분위기도 예쁘고. 여긴 독특하게 시간당으로...	조용하다/Adjective,골목/Noun,있다/Adjective,일단/Noun,너무...	4	1

감성분석 모델 학습을 위해 제거
- 별점이 없는 데이터
- 3점인 데이터

y_train, test 데이터 라벨링
- 별점 1,2점 / 0
- 별점 4,5점 / 1

01 RNN

1) 정수 인코딩

```
threshold = 3
total_cnt = len(tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('단어 집합(vocabulary)의 크기 :',total_cnt)
print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s'%(threshold - 1, rare_cnt))
print("단어 집합에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq / total_freq
```

```
단어 집합(vocabulary)의 크기 : 11692
등장 빈도가 2번 이하인 희귀 단어의 수: 6152
단어 집합에서 희귀 단어의 비율: 52.61717413616148
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 3.060536914807271
```

```
# 전체 단어 개수 중 빈도수 2이하인 단어 개수는 제거.
# 0번 패딩 토큰과 1번 OOV 토큰을 고려하여 +2
vocab_size = total_cnt - rare_cnt + 2
print('단어 집합의 크기 :',vocab_size)
```

```
단어 집합의 크기 : 5542
```

희귀 단어의 수와 등장 비율을 고려하여 단어 집합의 크기 확정
전체 11692개의 단어 집합(vocab_size)에서

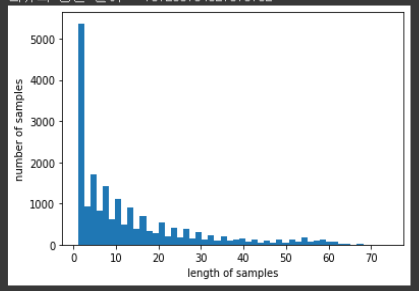
등장빈도가 2번 이하인 희귀 단어의 수의 비율이 52.6%, 실제 희귀 단어들의 등장 빈도는 3%

이기 때문에 이들을 모두 제외하고 정수 인코딩

2) 패딩

```
import matplotlib.pyplot as plt
print('리뷰의 최대 길이 :',max(len(l) for l in X_train))
print('리뷰의 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()
```

```
리뷰의 최대 길이 : 74
리뷰의 평균 길이 : 13.233154321613132
```




```
[ ] 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 = 30
    below_threshold_len(max_len, X_train)
```

```
전체 샘플 중 길이가 30 이하인 샘플의 비율: 87.52153207011855
```

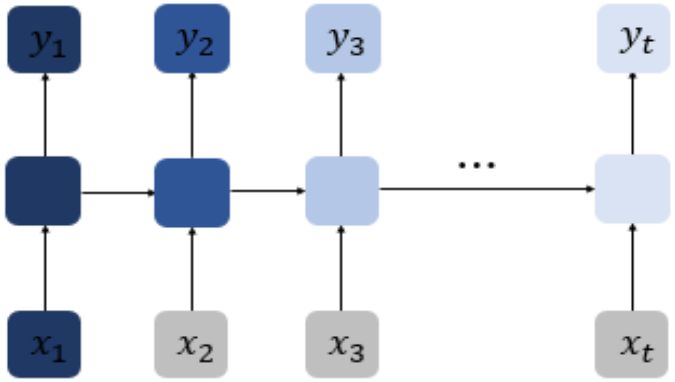
Neural Network의 텐서 연산을 위해 **리뷰의 평균 길이 및 길이별 리뷰의 분포를 고려**하여 모든 리뷰를 동일한 길이로 맞춰주는 작업

리뷰의 평균 길이는 13, 전체 데이터 셋에서 **길이가 30 이하인 샘플의 비율이 87%**인 점을 고려하여 모든 리뷰를 길이 30으로 패딩

01 RNN

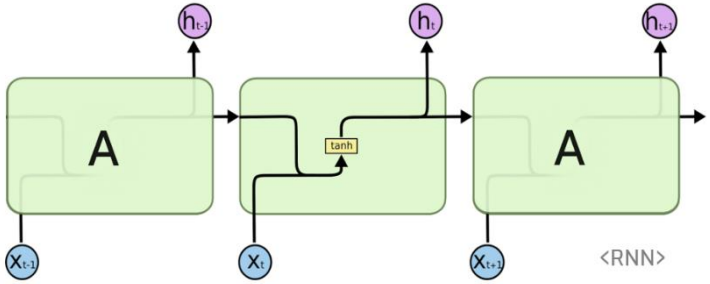
RNN의 고질적인 **gradient vanishing** 문제

인풋 값이 길수록, 즉 time step이 길어질수록 앞의 정보가 뒤로 충분히 전달되지 못하고 손실

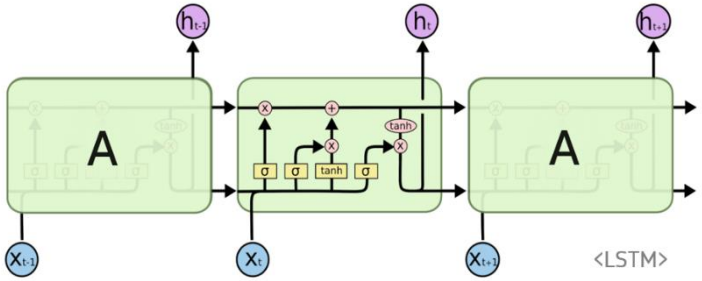


<감성분석을 위해 구현한 모델>

1) LSTM
2) Bi-LSTM
3) Bi-LSTM + Attention mechanism



Simple RNN은 **현재 시점 t의 인풋 Xt**와 **t-1 시점의 은닉층의 Ht-1**이 각각 가중치와 곱해져서 해당 RNN 셀의 입력이 되는 방식



LSTM의 경우 은닉층의 메모리 셀에 **Input Gate, Forget Gate, Output Gate**를 추가하고, 각 시점마다 이를 Cell state(셀 상태)에 누적시키는 구조

'Cell state'는 이전 노드에서 발생한 정보를 전달하는 일종의 **컨베이어 벨트**의 역할!

01 LSTM

```
from tensorflow.keras.layers import Embedding, Dense, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow import keras


model = Sequential()
model.add(Embedding(vocab_size, 100))  #임베딩 벡터의 차원 100
model.add(LSTM(128))
model.add(Dense(1, activation='tanh'))


es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('lstm_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc],
                    batch_size=60, validation_split=0.2)
```

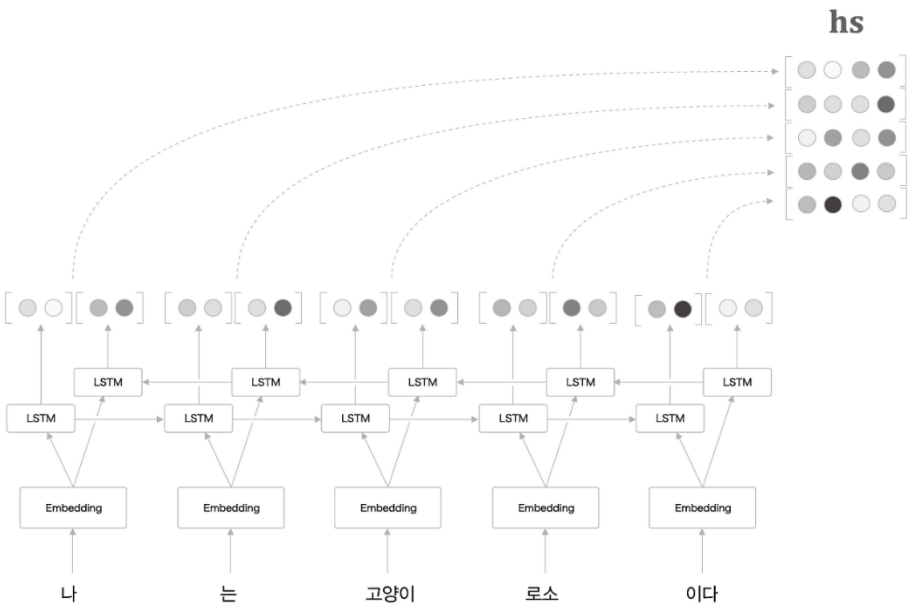
```
Epoch 1/15
264/264 [==============================] - 24s 84ms/step - loss: 0.7058 - acc: 0.7327 - val_loss: 0.2760 - val_acc: 0.9

Epoch 00001: val_acc improved from -inf to 0.92224, saving model to best_model_final.h5
Epoch 2/15
264/264 [==============================] - 21s 81ms/step - loss: 0.2635 - acc: 0.9148 - val_loss: 0.2384 - val_acc: 0.9

Epoch 00002: val_acc did not improve from 0.92224
Epoch 3/15
264/264 [==============================] - 21s 81ms/step - loss: 0.1758 - acc: 0.9460 - val_loss: 0.2996 - val_acc: 0.9

Epoch 00003: val_acc improved from 0.92224 to 0.93262, saving model to best_model_final.h5
Epoch 4/15
264/264 [==============================] - 21s 81ms/step - loss: 0.1472 - acc: 0.9630 - val_loss: 0.3315 - val_acc: 0.9

Epoch 00004: val_acc did not improve from 0.93262
```

```
loaded_model = load_model('lstm_model.h5')
print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, np.array(y_test))[1]))
```

```
155/155 [==============================] - 3s 13ms/step - loss: 0.3144 - acc: 0.9161

 테스트 정확도: 0.9161
```

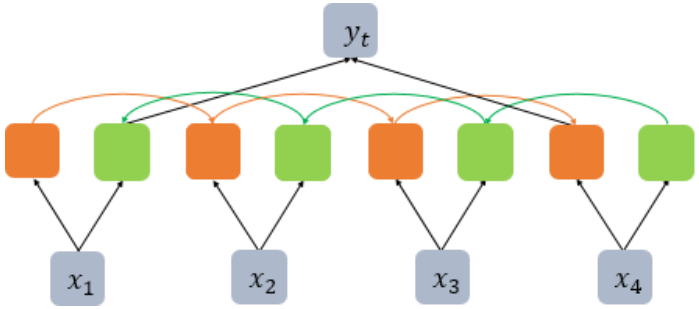
“ Accuracy : 91.16% ”

01 Bi-LSTM



양방향 LSTM은 **기존에 정방향으로 진행되는 LSTM 계층**에 **역방향으로 처리하는 LSTM 계층**을 추가하고,

두 LSTM 계층의 은닉 상태를 concatenate 하고, 히든 레이어를 거쳐 출력하는 구조



다-대-일(Many-to-one) 문제인 감성분석에서는 정방향 LSTM 기준에서의 마지막 시점의 은닉상태와,

역방향 LSTM 기준에서의 마지막 시점의 은닉 상태 즉, **첫번째 셀과 마지막 셀의 은닉 상태를 반환하고 이를 concatenation**

01 Bi-LSTM

BiLSTM 감성분석 모델링

```
[26] import re
     from tensorflow.keras.layers import Embedding, Dense, LSTM, Bidirectional
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.models import load_model
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
[314] model = Sequential()
      model.add(Embedding(vocab_size, 60))
      model.add(Bidirectional(LSTM(25)))
      model.add(Dense(1, activation='sigmoid'))
```

```
[315] es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
      mc = ModelCheckpoint('BiLSTM_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
```

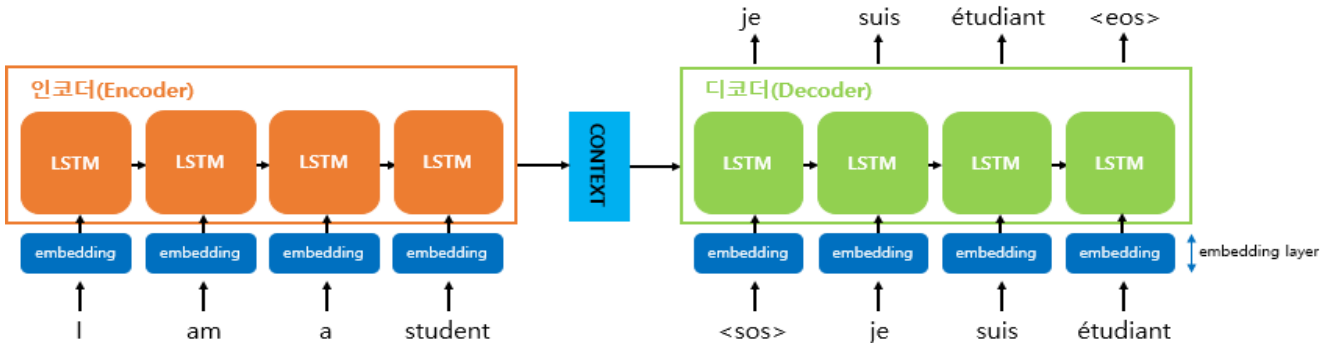
```
[316] model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
      history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=256, validation_split=0.20)

      Epoch 1/15
      62/62 [==============================] - 5s 22ms/step - loss: 0.5464 - acc: 0.7484 - val_loss: 0.2580 - val_acc: 0.9131

      Epoch 00001: val_acc improved from -inf to 0.91312, saving model to BiLSTM_model.h5
      Epoch 2/15
      62/62 [==============================] - 1s 10ms/step - loss: 0.2247 - acc: 0.9230 - val_loss: 0.2072 - val_acc: 0.9296

      Epoch 00002: val_acc improved from 0.91312 to 0.92958, saving model to BiLSTM_model.h5
      Epoch 3/15
      62/62 [==============================] - 1s 10ms/step - loss: 0.1723 - acc: 0.9394 - val_loss: 0.2071 - val_acc: 0.9230

      Epoch 00003: val_acc did not improve from 0.92958
      Epoch 4/15
      62/62 [==============================] - 1s 11ms/step - loss: 0.1342 - acc: 0.9523 - val_loss: 0.2118 - val_acc: 0.9184

      Epoch 00004: val_acc did not improve from 0.92958
```

```
loaded_model = load_model('BiLSTM_model.h5')
print("테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))
```

```
155/155 [==============================] - 1s 3ms/step - loss: 0.2014 - acc: 0.9301
테스트 정확도: 0.9301
```

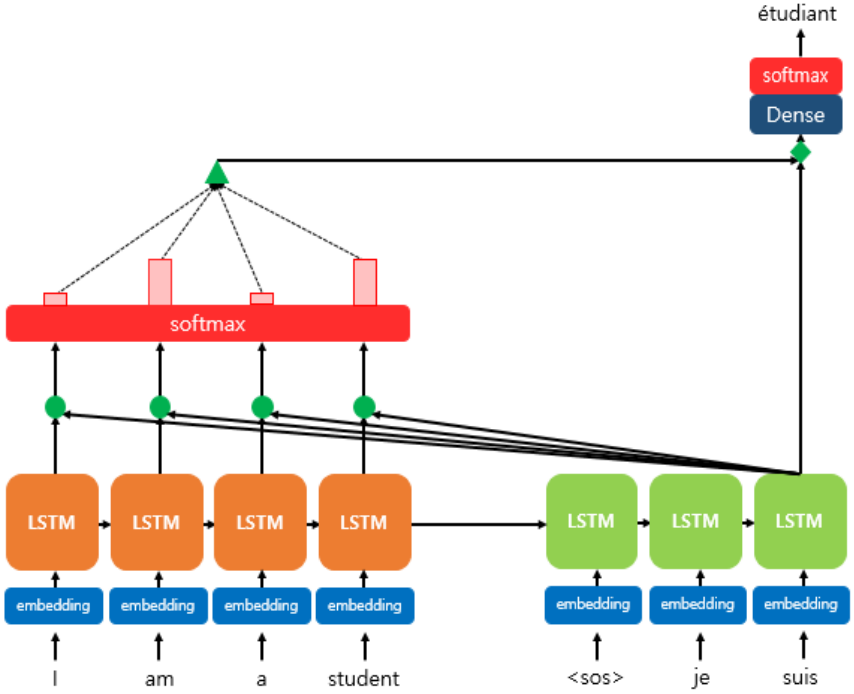
“ **Accuracy : 93.01%** ”

01 Bi-LSTM + Attention Mechanism



Rnn의 seq2seq 구조 : **하나의 고정된 크기의 벡터(Context)**에 모든 정보를 압축하기 때문에

정보 손실이 발생하는 seq2seq 구조적 문제점



1) 현재 Decoder의 시점 t에서 단어를 예측하기 위해, 인코더의 모든 은닉 상태 각각을 현재 시점의 은닉상태 St를 고려하여 Attention score를 구함

2) 이 모든 은닉 상태들 각각의 Attention score에 softmax함수를 취해 Attention Distribution(어텐션 분포)를 구함

3) 각 인코더의 어텐션 가중치와 은닉상태를 가중합하여 Attention Value(어텐션 값)을 구하고,

4) 어텐션 값과 디코더의 은닉상태를 concatenate하여 출력층의 연산에 반영 되도록 하는 매커니즘

01 Bi-LSTM + Attention Mechanism

ybigta

바다나우 Attention

```
import tensorflow as tf
```

```
[25] class BahdanauAttention(tf.keras.Model):
       def __init__(self, units):
         super(BahdanauAttention, self).__init__()
         self.W1 = Dense(units)
         self.W2 = Dense(units)
         self.V = Dense(1)

       def call(self, values, query): # 단, key와 value는 같음
         # query shape == (batch_size, hidden size)
         # hidden_with_time_axis shape == (batch_size, 1, hidden size)
         # score 계산을 위해 뒤에서 할 덧셈을 위해서 차원을 변경
         hidden_with_time_axis = tf.expand_dims(query, 1)

         # score shape == (batch_size, max_length, 1)
         # we get 1 at the last axis because we are applying score to self.V
         # the shape of the tensor before applying self.V is (batch_size, max_length, units)
         score = self.V(tf.nn.tanh(
             self.W1(values) + self.W2(hidden_with_time_axis)))

         # attention_weights shape == (batch_size, max_length, 1)
         attention_weights = tf.nn.softmax(score, axis=1)

         # context_vector shape after sum == (batch_size, hidden_size)
         context_vector = attention_weights * values
         context_vector = tf.reduce_sum(context_vector, axis=1)

         return context_vector, attention_weights
```

감성분석 모델은 엄밀히 따지면 전체 문장의 감성을 0~1사이의 값으로 표현하는 seq2vec이지만, attention의 mechanism 자체는 동일

마지막 은닉상태의 값이 감성분석한 점수를 나타내고, LSTM 셀을 거치며 손실되었지만 유의미한 정보들에 attention value를 가중하여 분석의 Accuracy를 높이는 방식

$$score(query,\ key) = V^T tanh(W_1 key + W_2 query)$$

바다나우(Bahdanau) 어텐션 : 어텐션 값을 구하는 수식만 다를 뿐, 위에서 언급한 닷 프로덕트 어텐션과 전체적인 메커니즘은 동일

01 Bi-LSTM + Attention Mechanism

BiLSTM + Attention 감성분석 모델링

```
[38] sequence_input = Input(shape=(max_len,), dtype='int32')
     embedded_sequences = Embedding(vocab_size, 128, input_length=max_len, mask_zero = True)(sequence_input)

[39] lstm = Bidirectional(LSTM(64, dropout=0.5, return_sequences = True))(embedded_sequences)

[40] lstm, forward_h, forward_c, backward_h, backward_c = Bidirectional #
       (LSTM(64, dropout=0.5, return_sequences=True, return_state=True))(lstm)

[41] print(lstm.shape, forward_h.shape, forward_c.shape, backward_h.shape, backward_c.shape)

     (None, 30, 128) (None, 64) (None, 64) (None, 64) (None, 64)

[42] state_h = Concatenate()([forward_h, backward_h]) # 은닉 상태
     state_c = Concatenate()([forward_c, backward_c]) # 셀 상태

[43] attention = BahdanauAttention(64) # 가중치 크기 정의
     context_vector, attention_weights = attention(lstm, state_h)

[44] dense1 = Dense(20, activation="relu")(context_vector)
     dropout = Dropout(0.5)(dense1)
     output = Dense(1, activation="tanh")(dropout)
     model = Model(inputs=sequence_input, outputs=output)

[45] es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
     mc = ModelCheckpoint('BiLSTM_Attention_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

[46] model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

[47] history = model.fit(X_train, y_train, epochs = 4, callbacks=[es, mc], batch_size = 128, validation_split=0.2)

     Epoch 1/4
     124/124 [==============================] - 40s 218ms/step - loss: 2.1244 - accuracy: 0.3766 - val_loss: 0.2819 - val_accuracy: 0.8954
     WARNING:tensorflow:Can save best model only with val_acc available, skipping.
     Epoch 2/4
     124/124 [==============================] - 23s 187ms/step - loss: 0.2637 - accuracy: 0.9075 - val_loss: 0.2048 - val_accuracy: 0.9222
     WARNING:tensorflow:Can save best model only with val_acc available, skipping.
     Epoch 3/4
     124/124 [==============================] - 23s 187ms/step - loss: 0.1876 - accuracy: 0.9345 - val_loss: 0.2093 - val_accuracy: 0.9278
     WARNING:tensorflow:Can save best model only with val_acc available, skipping.
```

```
print("₩n 테스트 정확도: %.4f" % (model.evaluate(X_test, y_test)[1]))

155/155 [==============================] - 4s 25ms/step - loss: 0.2201 - accuracy: 0.9260

 테스트 정확도: 0.9260
```

“ Accuracy : 92.60% ”

Bi-LSTM + Attention mechanism과 처음 사용한 LSTM 모델의 Accuracy에서 큰 차이가 없는 이유?

1) 한정된 데이터 셋의 크기

2) 리뷰의 특성상 그렇게 복잡한 의미를 함축한 긴 문장이 아니기 때문

정확도에서 유의미한 차이를 보이지 않았기 때문에 Attention mechanism을 적용한 모델보다 상대적으로 가벼운 Bi-LSTM을 최종적으로 감성분석 모델로 채택

<데이터셋 관련 요인들 간의 비교>

200,000건의 네이버 영화 리뷰 데이터로 학습 -> 정확도 53%

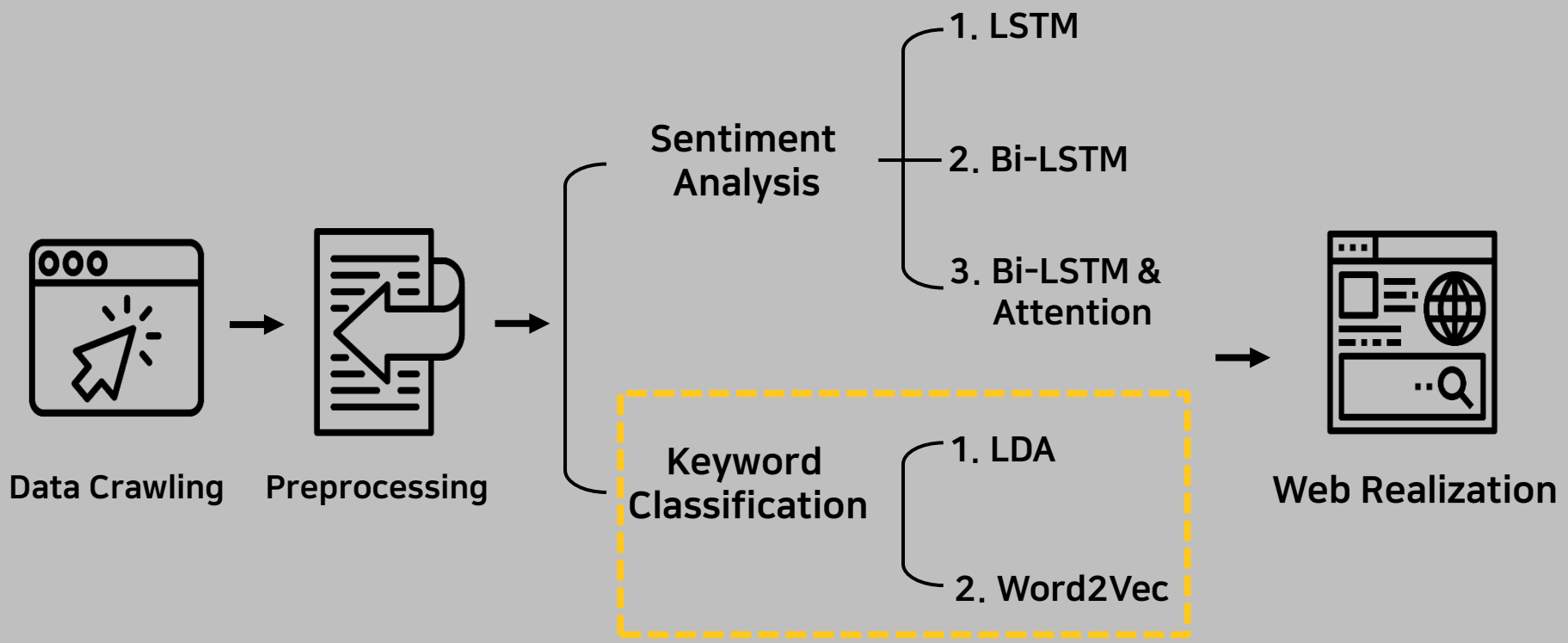
약 12,000건의 리뷰 데이터로 학습 -> 정확도 84%

약 35,000건의 리뷰 데이터로 학습 -> 정확도 93%

➔ 모델의 목적에 맞는 적합한 훈련용 데이터를 사용하는 것과, 데이터셋의 절대적인 크기의 중요성

추가로 시간적 여건으로 **JMT 등과 같은 외국어**와, **'아바라'와 같은 줄임말 토큰**들은 모두 전처리 과정에서 지워지거나, 본래의 의미를 갖지 못하게 되었지만, 이러한 토큰들 역시 문장의 감성과 의미를 함축한 중요한 요소이기에 이러한 부분들을 고려하여 전처리가 조금 더 세심하게 이뤄진다면 정확도가 더 높아질 것으로 기대

03 키워드 분류 모델


Data Crawling
Preprocessing
Sentiment Analysis
1. LSTM
2. Bi-LSTM
3. Bi-LSTM & Attention
Keyword Classification
1. LDA
2. Word2Vec
Web Realization

02 LDA + LSTM

[LDA 토픽 모델링과 LSTM을 활용한 키워드 분류]

해당 리뷰의 키워드를 나타내는 레이블 無

Review	Keyword
연말이라 분위기 좋은 카페를 찾고있었는데, 외관도 그렇고 카페 전반적인 분위기가 너무 좋습니다! 약간 고급스러운(?) 크리스마스 느낌 물씬..	
여기 라떼 계속 생각나요... 맛이 여운이 남는달까	
알바들 다 물갈이해야됨 인성 답없음	
프랑스의 분위기를 느낄 수 있었어요 근데 화장실까지 프랑스라 좀 더러워서 1점 감점해요.	
직원분들 매우 친절하심... 맛은 보통	
수플레는 괜찮은데 딸기가 너무 별로ㅠ 블루베리는 괜찮았음	

Train Data for LSTM

LDA

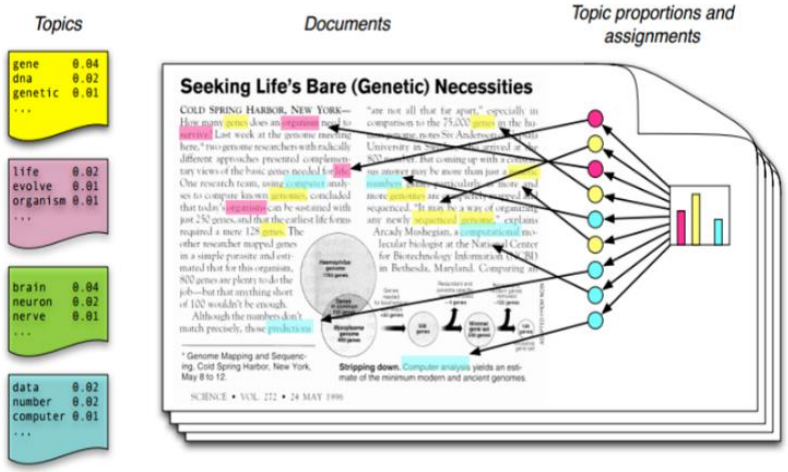
Review	Keyword
연말이라 분위기 좋은 카페를 찾고있었는데, 외관도 그렇고 카페 전반적인 분위기가 너무 좋습니다! 약간 고급스러운(?) 크리스마스 느낌 물씬..	분위기
여기 라떼 계속 생각나요... 맛이 여운이 남는달까	커피/디저트
알바들 다 물갈이해야됨 인성 답없음	서비스
프랑스의 분위기를 느낄 수 있었어요 근데 화장실까지 프랑스라 좀 더러워서 1점 감점해요.	분위기
직원분들 매우 친절하심... 맛은 보통	서비스
수플레는 괜찮은데 딸기가 너무 별로ㅠ 블루베리는 괜찮았음	커피/디저트

LSTM

"새로 입력된 사용자 리뷰" → [KeyWord]

비지도학습 텍스트 마이닝 기법 중에 하나인 **LDA(Latent Dirichlet Allocation)** 토픽 모델링을 통해 각각의 리뷰에 키워드 레이블링

LDA를 통해 **라벨링된 리뷰 데이터를 학습 데이터로 활용**하여 LSTM 모델 구현)



LDA란?

문서 내 토픽 분포와 **각 토픽에 포함된 단어의 확률분포**를 디리클레 분포로 가정하여 각 문서에 어떤 토픽들이 존재하는지 찾아내는 **비지도학습 알고리즘**

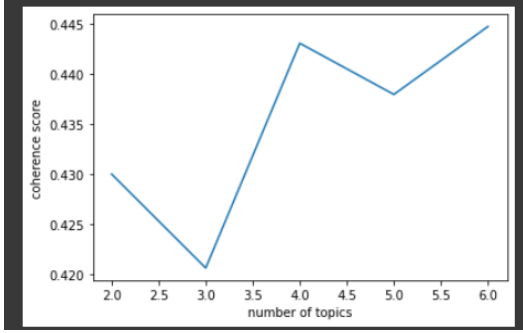
최적의 토픽 수 찾기

LDA를 이용해 토픽 모델링을 하기 위해 **토픽 수(num_topics)를 하이퍼파라미터로 설정**

Perplexity(혼잡도), Coherence(유사도)를 기반으로 토픽의 수 설정

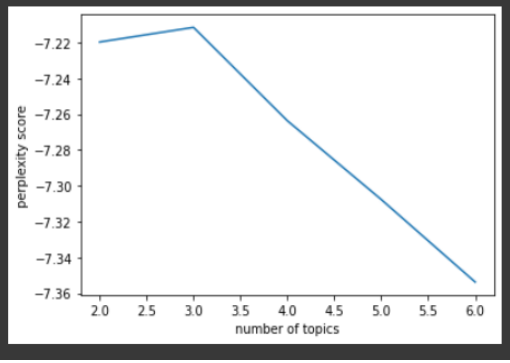
```
import matplotlib.pyplot as plt
%matplotlib inline

coherence_values=[]
for i in range(2,7):
  ldamodel = gensim.models.LdaModel(corpus, num_topics = i, id2word=dictionary, passes = 100)
  coherence_model_lda = CoherenceModel(model=ldamodel, texts=corpus_gen, dictionary=dictionar
  coherence_lda = coherence_model_lda.get_coherence()
  coherence_values.append(coherence_lda)
```

```
x=range(2,7)
plt.plot(x, coherence_values)
plt.xlabel("number of topics")
plt.ylabel("coherence score")
plt.show()
```




```
import matplotlib.pyplot as plt
%matplotlib inline

perplexity_values=[]
for i in range(2,7):
  ldamodel = gensim.models.LdaModel(corpus, num_topics = i, id2word=dictionary, passes = 100)
  perplexity_values.append(ldamodel.log_perplexity(corpus))
```

```
x=range(2,7)
plt.plot(x, perplexity_values)
plt.xlabel("number of topics")
plt.ylabel("perplexity score")
plt.show()
```

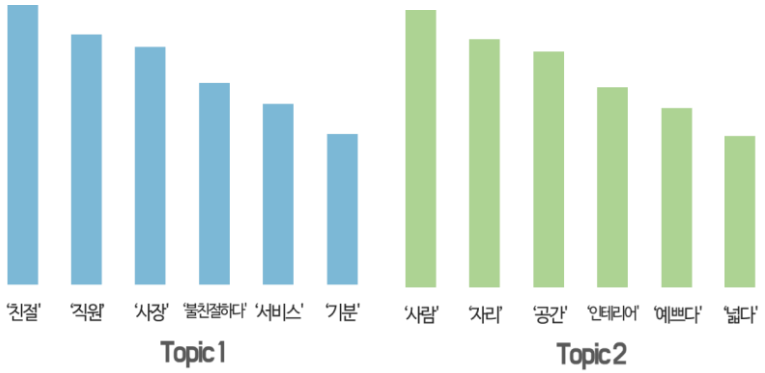


Perplexity 각각의 단어들이 여러 토픽에 걸쳐서 분포하는 정도 → 낮을수록 토픽 수가 적절하다는 의미

Coherence 주제의 일관성을(각 토픽의 상위 10개 단어간 유사도를 측정) → 높을수록 각 토픽의 의미론적 일관성이 높다는 것을 의미

토픽 개수의 증가에 따른 coherence와 perplexity 그래프를 통해 이 둘을 고려하여 '4'로 num_topics 설정

02 LDA + LSTM



→ '서비스'

→ '분위기'

Review	Keyword
연말이라 분위기 좋은 카페를 찾고있었는데, 외관도 그렇고 카페 전반적인 분위기가 너무 좋습니다! 약간 고급스러운(?) 크리스마스 느낌 물씬..	분위기
여기 라떼 계속 생각나요... 맛이 여운이 남는달까	커피/디저트
알바들 다 물갈이해야됨 인성 답없음	서비스
프랑스의 분위기를 느낄 수 있었어요 근데 화장실까지 프랑스라 좀 더러워서 1점 감점해요.	분위기
직원분들 매우 친절하심... 맛은 보통	서비스
수플레는 괜찮은데 딸기가 너무 별로ㅠ 블루베리는 괜찮았음	커피/디저트

Multilabel classification LSTM 모델

LDA를 통해 데이터를 라벨링한 후, 이를 학습 데이터로 사용하여 인풋 리뷰의 토픽을 예측하는 Multilabel classification LSTM 모델 구현

“ Accuracy : 89.97% ”

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

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

history = model.fit(X_train, y_train, batch_size=128, epochs=30, callbacks=[es, mc], validation_data=(X_test, y_test))
```

```
Epoch 1/30
148/148 [==============================] - 7s 15ms/step - loss: 1.1871 - acc: 0.4649 - val_loss: 0.5616 - val_acc: 0.

Epoch 00001: val_acc improved from -inf to 0.79511, saving model to best_model.h5
Epoch 2/30
148/148 [==============================] - 2s 12ms/step - loss: 0.4047 - acc: 0.8664 - val_loss: 0.3189 - val_acc: 0.

Epoch 00002: val_acc improved from 0.79511 to 0.88417, saving model to best_model.h5
Epoch 3/30
148/148 [==============================] - 2s 12ms/step - loss: 0.1726 - acc: 0.9455 - val_loss: 0.3398 - val_acc: 0.

Epoch 00003: val_acc improved from 0.88417 to 0.89033, saving model to best_model.h5
Epoch 4/30
148/148 [==============================] - 2s 12ms/step - loss: 0.1461 - acc: 0.9532 - val_loss: 0.3333 - val_acc: 0.

Epoch 00004: val_acc improved from 0.89033 to 0.89309, saving model to best_model.h5
Epoch 5/30
148/148 [==============================] - 2s 12ms/step - loss: 0.0670 - acc: 0.9799 - val_loss: 0.3298 - val_acc: 0.

Epoch 00005: val_acc improved from 0.89309 to 0.89968, saving model to best_model.h5
Epoch 6/30
148/148 [==============================] - 2s 12ms/step - loss: 0.0582 - acc: 0.9825 - val_loss: 0.3515 - val_acc: 0.

Epoch 00006: val_acc did not improve from 0.89968
Epoch 00006: early stopping
```

```
loaded_model = load_model('best_model.h5')
print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))
```

```
148/148 [==============================] - 1s 3ms/step - loss: 0.3298 - acc: 0.8997

 테스트 정확도: 0.8997
```

Topic 3: '맛있다', '커피', '디저트', '분위기', '비싸다', '가격', '딸기', 음료' → ??

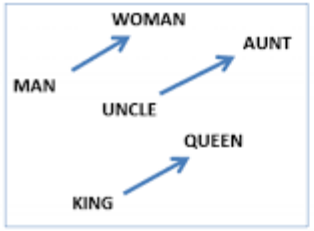
Topic 4: '빙수', '맛있다', '여기', '커피', '디저트', '시간', '좋아하다', '맛없다' → ??

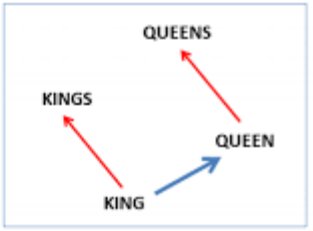
LDA를 활용한 토픽 분류의 한계점

1) LDA가 확률 기반의 분류이기 때문에 생각보다 사용자 입장에서 유의미한 토픽으로 잘 분류되지 않음

2) 토픽들 간에 차이가 극명하게 드러나지 않음

3) 리뷰 한문장에 포함된 단어가 여러 토픽에 걸쳐 분포한 경우에도 토픽 분류에 어려움

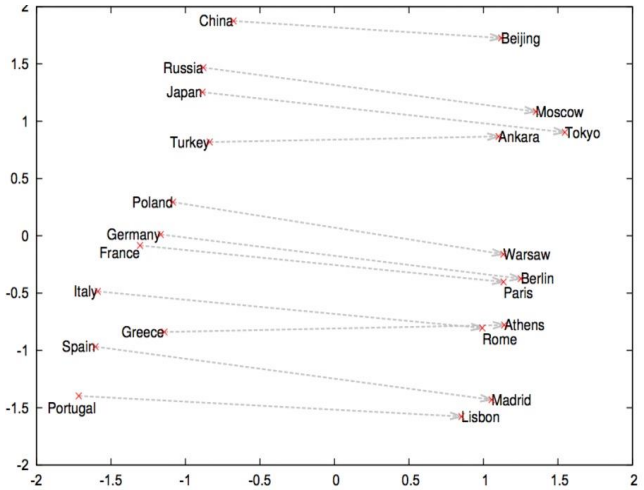




Word2Vec를 이용한 키워드 분석

Word2Vec의 정의: "주변의 단어가 비슷하면 해당 단어가 비슷하다"고 가정하고 다차원의 벡터 공간 상에 단어들 간의 관계(유사도)를 고려하여 표현하는 것

우리가 원하는 키워드를 직접 뽑아 토픽을 분류하고자 하는 프로젝트의 목적에 더 적합



Word2Vec 모델 생성

```
from gensim.models import Word2Vec
model = Word2Vec(sentences = X, size = 100, window = 3, min_count = 20, workers = 4
, sg = 0, iter=200)
```

벡터의 차원, context word의 범위를 정하는 window, 단어 최소 빈도 등을 설정하고 word2vec의 두가지 방식 중 skip-gram을 선택하여 모델 설정

워드벡터 생성(Word Embedding)

	0	1	2	...	96	97	98	99
크루아상	0.064190	-0.527391	0.131361	···	-0.624361	1.598588	-1.191401	-1.709759
디자인	-0.752420	0.339741	-1.335791	···	0.407871	0.638819	-2.131828	-0.118566
라떼	-1.036351	-0.821037	-1.256901	···	0.449643	-0.269316	-0.780564	0.750244
인스타	0.174230	-0.801964	-1.111442	···	0.547558	-1.322446	0.001137	1.500044
				⋮				
딸기	-2.251453	-0.165115	-0.099334	···	0.406585	-0.299693	-0.431127	1.447366

* 모든 리뷰데이터 내에 포함된 단어들 중 20회 이상의 빈도를 지닌 단어들을 100차원 벡터로 표현

Genism 라이브러리가 제공하는 most_similar()를 이용해서 **단어 간 유사도** 확인

```
- print(model.wv.most_similar("인테리어/Noun"))
```
: 분위기, 디자인, 공간, 모던, 콘셉트, 꾸미다, 아기자기하다, 아늑하다, 어둡다

```
- print(model.wv.most_similar("에스프레소/Noun"))
```
: 라떼, 커피, 아메리카노, 핫초코, 진하다, 아이스, 우유, 원두, 코코넛, 필터

```
- print(model.wv.most_similar("불친절하다/Adjective"))
```
: 불친절, 태도, 불쾌하다, 싹수없다, 교육, 싸가지, 무섭다, 말투, 친절하다

```
- print(model.wv.most_similar("마카롱/Noun"))
```
: 케이크, 필링, 디저트, 스콘, 타르트, 크루아상, 치즈케이크, 꼬끄

02 Word2vec

거리행렬(Distance Matrix)

	크루아상	디자인	라떼	인스타	...	분위기	케이크	딸기
크루아상	2.487420e-17	4.285202e-19	2.783494e-18	6.428339e-18		6.497285e-21	5.445805e-18	2.487420e-17
디자인	3.358046e-18	1.155579e-20	2.340290e-19	1.472308e-19		4.777407e-19	8.721168e-21	1.264427e-16
라떼	1.147857e-16	2.660540e-18	1.576986e-18	7.802928e-18		4.533370e-18	1.484832e-16	1.432694e-18
인스타	4.590693e-18	1.459370e-17	5.673584e-22	1.662715e-18		3.435373e-18	1.107346e-17	9.367931e-17
					⋮			
딸기	1.107346e-17	9.367931e-17	6.480159e-20	2.814873e-20		8.015857e-21	1.925283e-19	7.305931e-21

```python
# 유클리디안 거리 구하기!
def euclidian_dist(p1,p2):
    dist = np.linalg.norm(p1 - p2)
    return dist
```

유클리디안 거리로 단어간 거리(유사도) 표현

거리벡터에서 키워드로 사용할 단어만 추출

	token	커피	가격	직원	분위기
0	좋다	1.44055e-12	5.69393e-18	1.58470e-15	3.05779e-11
1	맛	3.40663e-11	3.60404e-15	1.59036e-16	1.28656e-13
2	커피	1.00000e+00	9.66869e-16	8.93167e-17	6.08449e-12
...	...	...	...	...	...
1087	선곡	5.57393e-19	1.37287e-22	7.65590e-20	6.57060e-15

WordCloud 시각화를 통한 대표 쿼리 단어 4개 선정

각 리뷰의 **키워드별 스코어** 구하기 → 각 키워드의 거리 행렬을 이용해서 **문장별 키워드 스코어 계산**

	token	커피	가격	직원	분위기	
0	좋다	1.44055e-12	5.69393e-18	1.58470e-15	3.05779e-11	* 1
1	맛	3.40663e-11	3.60404e-15	1.59036e-16	1.28656e-13	* 0
2	커피	1.00000e+00	9.66869e-16	8.93167e-17	6.08449e-12	* 0
...	...	...	...	...	...	
1087	선곡	5.57393e-19	1.37287e-22	7.65590e-20	6.57060e-15	* 1

$$\begin{matrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\ f_1 & 1.1 & 0.2 & 0.1 & 0.7 & 0.8 & 0.6 & 0.4 \\ f_2 & 0.8 & 0.5 & 0.8 & 1.0 & 0.8 & 0.2 & 0.1 \\ f_3 & 0.2 & 0.3 & 0.5 & 0.6 & 0.2 & 0.1 & 0.7 \end{matrix} \times \begin{matrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{matrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{matrix} = \begin{matrix} 0.8 \\ 0.7 \\ 0.4 \end{matrix}$$

'선곡이 좋아요' → [선곡] + [좋다] →

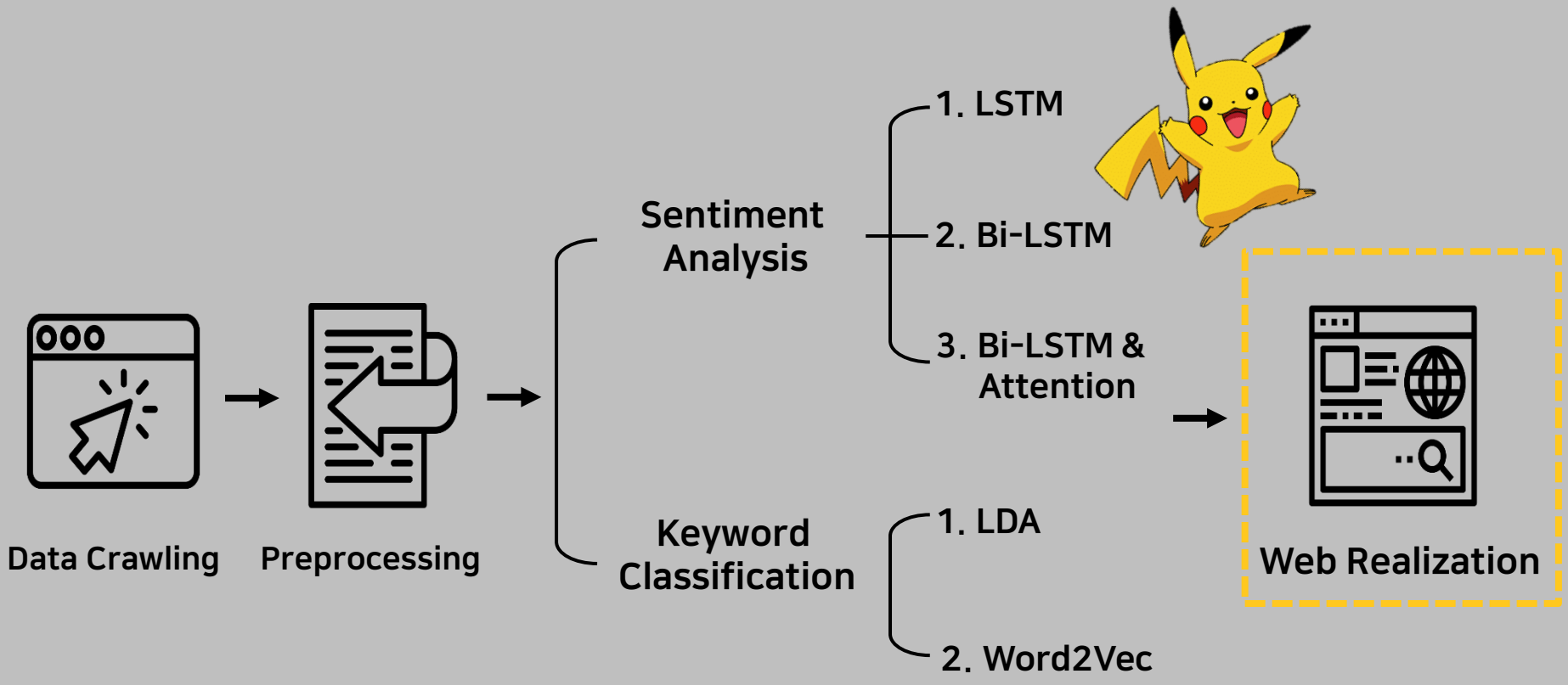
'분위기' score = 6.57060e-15 * 1 + (1.28656e-13 * 0 + 6.08449e-12 * 0 + ⋯) + 3.05779e-11 * 1

(계산상 효율성을 위해 **단어문서행렬**을 만들어 **가중치행렬과 내적곱**을 해주는 방식으로 스코어 산출)

	token	커피	가격	직원	분위기
0	좋다	1.44055e-12	5.69393e-18	1.58470e-15	3.05779e-11
1	맛	3.40663e-11	3.60404e-15	1.59036e-16	1.28656e-13
2	커피	1.00000e+00	9.66869e-16	8.93167e-17	6.08449e-12
...	...	...	...	...	...
1087	선곡	5.57393e-19	1.37287e-22	7.65590e-20	6.57060e-15
		1.4405e-12	5.6905e-18	1.5848e-15	3.0578e-11

'선곡이 좋다'는 예시 리뷰에 대해 분위기 스코어가 네 스코어 중 가장 높음 → **keyword = '분위기'**

04 웹 시연


Data Crawling
Preprocessing
Sentiment Analysis
1. LSTM
2. Bi-LSTM
3. Bi-LSTM & Attention
Keyword Classification
1. LDA
2. Word2Vec
Web Realization

감사합니다.

https://github.com/jiyoung98/ybigta_project

Q&A