Stress Detection
on Social Media Posts

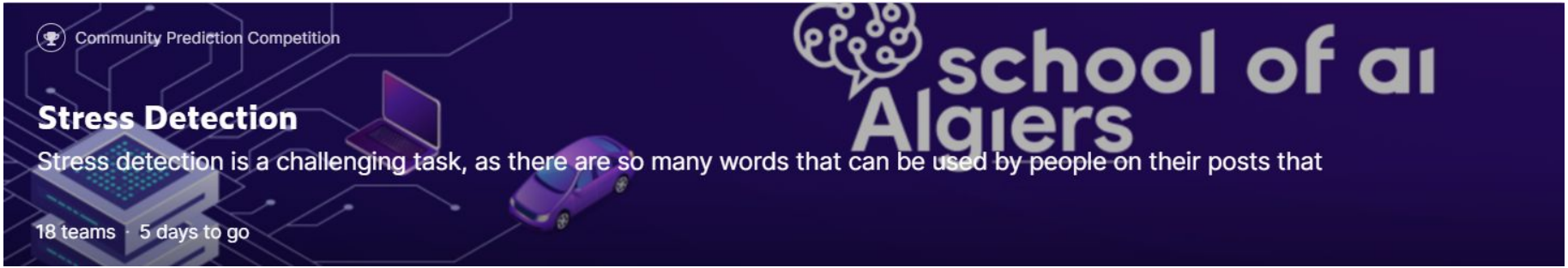
OB TEAM 2

Table of contents

01

EDA

-Loading datasets
-Wordcloud visualization



Goal: text와 label column을 사용하여 스트레스를 탐지

- 116개의 칼럼의 데이터 세트
- 정신 건강과 관련된 subreddit에 게시된 텍스트

train 데이터

- train 데이터셋의 개수 1986개
- 칼럼수 116개
- ‘text, ‘label´ 칼럼만 사용해야 한다고 대회 안내에 언급

	post_id	subreddit	post_id.1	sentence_range	text	id	label	confidence	social_timestamp	social_karma
0	2092	stress	8b9j8h	(0, 5)	Hello, You are invited to complete a survey fo...	6319	0	1.000000	1523381550	1
1	1567	homeless	8cprrd	(2, 7)	I want to know about your story, how you got t...	42996	0	0.800000	1523903160	1
2	1550	ptsd	63nmo1	(5, 10)	Have any of you requested accommodations throu...	34744	0	1.000000	1491418808	7
3	2785	ptsd	9vkbbr	[0, 5]	Whenever I have a reaction or meltdown, he alw...	614	1	0.571429	1541769275	3
4	1936	relationships	7pdcp5	(0, 5)	Throwaway because this is an embarrassing issu...	14399	1	0.800000	1515560629	1

5 rows × 117 columns

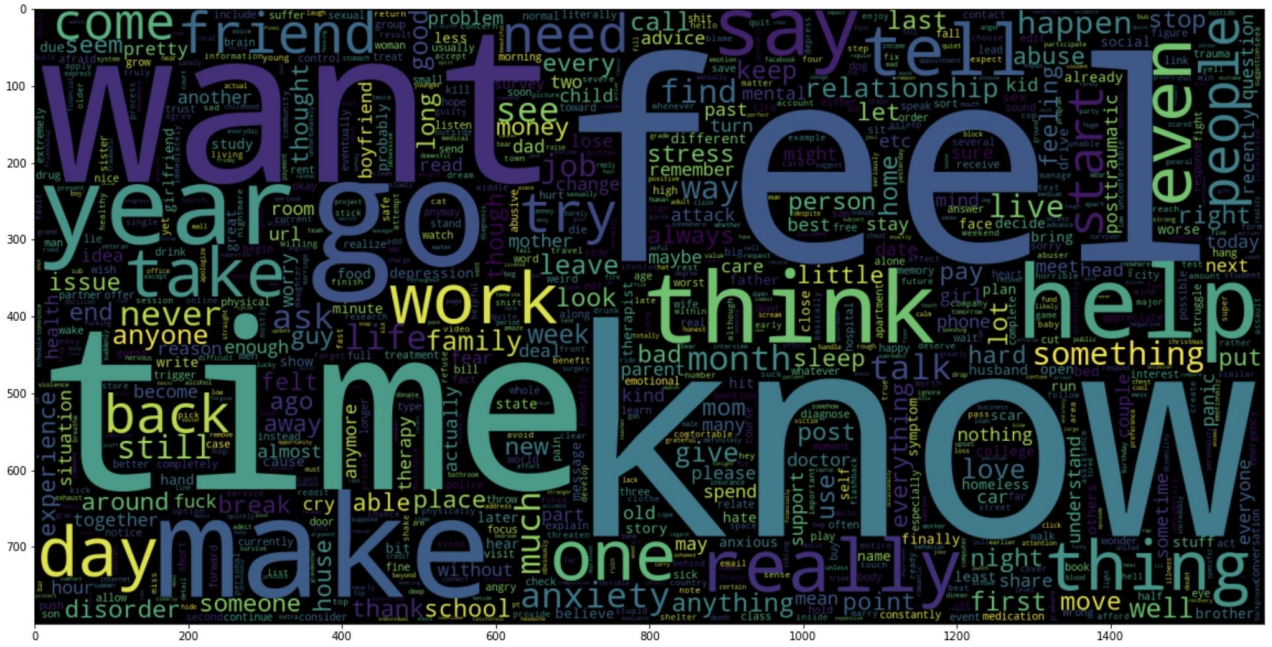
text

- 정신건강과 관련있는 하위레딧[reddit]의 게시글
- 사람들이 공유하는 다양한 정신 건강 관련 경험이 포함된 텍스트

label

- 스트레스 여부에 따라 1[stress], 0[no stress]로 구분

wordcloud 결과
전반적으로 know, feel, want, think 등 인지과정과 관련있는 동사가 자주 등장

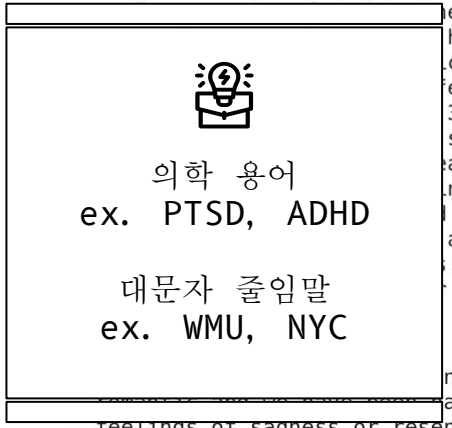


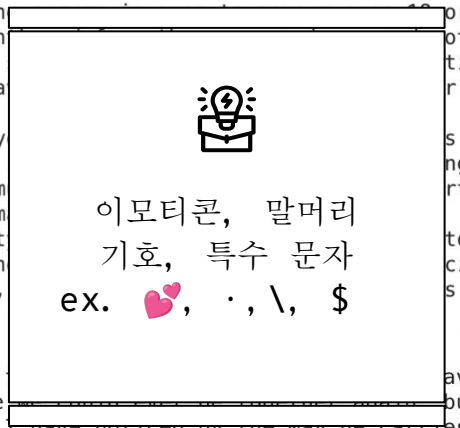
```
[ ] all_data.text.tolist()
```

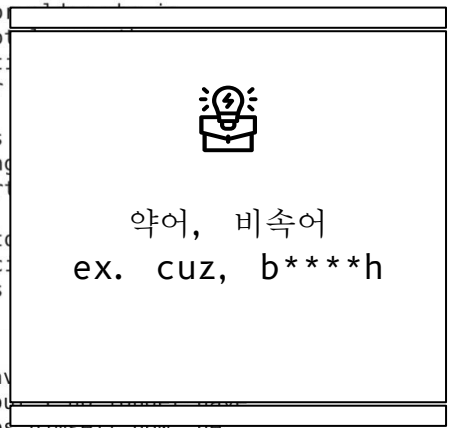
```
['Hello, You are invited to complete a survey for a WMU psychology department research project designed to assess
treatment preferences among adults seeking treatment for a variety of concerns. We hope to learn if and how
preferences for treatment change over time, and if patient’s perception of their treatment’s match to their
preferences is related to the benefit received from treatment. The survey is open to anyone ages 18 or older who is
currently engaged in mental health treatment for at least one month and four therapy sessions, and not longer than one
year. If you choose to participate, you will be asked to provide some demographic/background information, respond to
survey items about your preferences for treatment and how they have changed over time, and about your quality of life.
The survey may take between 30-40 minutes to complete.',
 "I want to know about your story, how you got there, struggles you never realized you would face (as well as how you
dealt with the ones you already expected), how you got out of it (if you did). Anything and everything you think might
be relevant for me to look into. You do not HAVE to be a singer/musician to reply. Every story is worth listening to
for me. If you're in NYC and can meet in person, that would be amazing as well.",
 "Have any of you requested accommodations through your university, and if so, would you be willing to share what's
helped/hasn't helped? Thanks! Edit: Thank you SO MUCH to those who have responded! Are there any specific
accommodations you asked for that were beneficial? (For instance, for my day job I work in classrooms with kids that
have ADHD.",


 'We are 21 now, both turning 22 next month. It started off as friendly catching up, but things have gotten more
romantic and we have been hanging out a lot more. I wasn’t sure we could ever be together again, but I no longer have
feelings of sadness or resentment regarding what he did to me. I have noticed by the way he carries himself now, he
has definitely matured. I guess I just want to know when or if I should tell him I hooked up with his friends?',
 'I have started working on a youtube series about coping with and healing PTSD using movement therapy to find flow
(hooping is my outlet). I am super excited to share this with you beautiful, strong, wonderful survivors, so please
share this with anyone else who you think could benefit from this series💕 Wishing you well! Until next time, may
peace love and light always be with you! [Journal Entry: January 18, <url> Edit: formatting',
 ' ---   **tl;dr**: Met this guy a few months ago, we\'ve only hung out a few times. I\'m a little wary of the age
gap and don\'t have much experience with accurately "reading signals". He is mildly physically affectionate (leans
towards me, hits my arm playfully, pulls things from my hair) and gives compliments that seem genuine and sincere, but
aren\'t quite non platonic. What do you think/what should I look for?',
```

```
[ ] all_data.text.tolist()
```

['Hello, You are invited to complete a survey for a WMU psychology department research project designed to assess treatment preferences among adults seeking treatment for a variety of concerns. We hope to learn if and how preferences for treatment change over time, and if patient’s perception of their treatment’s match to their







feelings of sadness or resentment regarding what he did to me. I have noticed by the way he carries himself now, he has definitely matured. I guess I just want to know when or if I should tell him I hooked up with his friends?',
'I have started working on a youtube series about coping with and healing PTSD using movement therapy to find flow (hooping is my outlet). I am super excited to share this with you beautiful, strong, wonderful survivors, so please share this with anyone else who you think could benefit from this series💕 Wishing you well! Until next time, may peace love and light always be with you! [Journal Entry: January 18, <url> Edit: formatting',
' --- **tl;dr**: Met this guy a few months ago, we\'ve only hung out a few times. I\'m a little wary of the age gap and don\'t have much experience with accurately "reading signals". He is mildly physically affectionate (leans towards me, hits my arm playfully, pulls things from my hair) and gives compliments that seem genuine and sincere, but aren\'t quite non platonic. What do you think/what should I look for?',

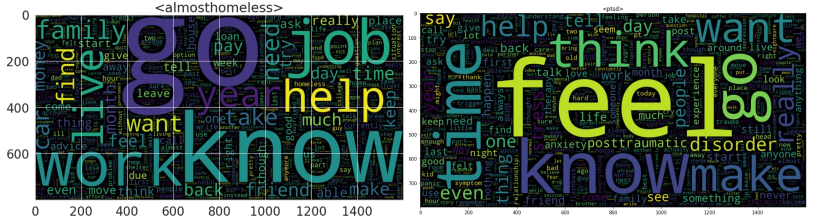
Subreddit 관련 추가 EDA

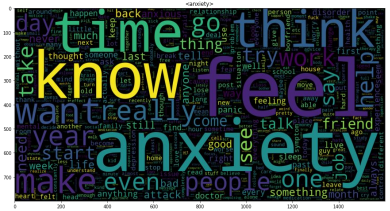
- 특정 subreddit 주제에 따라 키워드 차이가 크지 않음

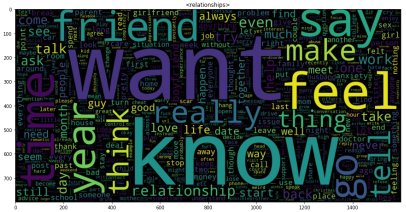
  [상위빈도 단어가 다른 것은 subreddit 제목 자체-homeless-인 경우]

- Subreddit 제목이 ‘stress’, ‘ptsd’ 등 스트레스를 포함할 때 라벨 1이 가장 많을까?

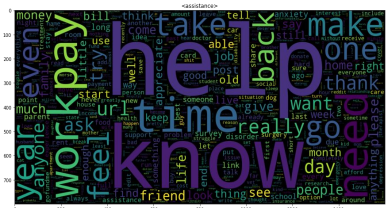
  : subreddit별 차이는 있지만 NO

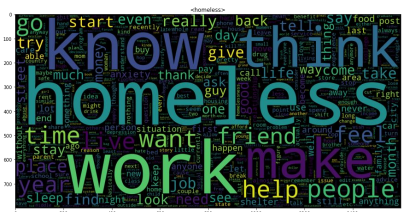


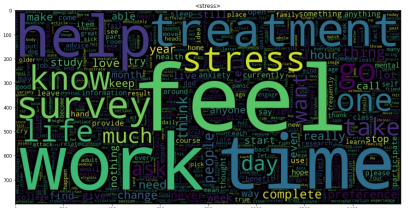


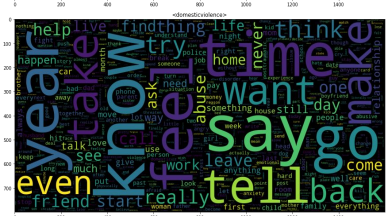


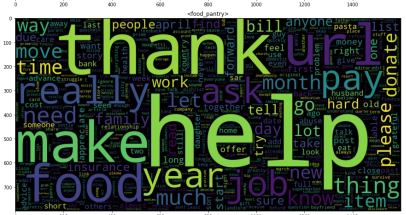
subreddit	
almosthomeless	0.673077
anxiety	0.678670
assistance	0.338095
domesticviolence	0.624434
food_pantry	0.291667
homeless	0.390244
ptsd	0.564356
relationships	0.456464
stress	0.521739
survivorsofabuse	0.487952

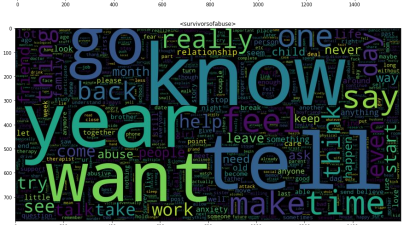












02

Text Cleansing

-combining train, test data
-adding sentiment column
-text preprocessing

```
data_all = pd.concat([train, test])
data_all.reset_index(drop=True, inplace=True)
data_all = data_all[['post_id','text']]
```

	post_id	text
0	2092	Hello, You are invited to complete a survey fo...
1	1567	I want to know about your story, how you got t...
2	1550	Have any of you requested accommodations throu...
3	2785	Whenever I have a reaction or meltdown, he alw...
4	1936	Throwaway because this is an embarrassing issu...

‘post_id’: kaggle 제출 시 필요
‘text’: 분석에 사용되는 칼럼

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

data_all['pos'] = data_all['text'].apply(lambda x: analyzer.polarity_scores(x)['pos'])
data_all['neg'] = data_all['text'].apply(lambda x: analyzer.polarity_scores(x)['neg'])
data_all['neu'] = data_all['text'].apply(lambda x: analyzer.polarity_scores(x)['neu'])
data_all['compound'] = data_all['text'].apply(lambda x: analyzer.polarity_scores(x)['compound'])
```

	post_id	text	pos	neg	neu	compound
0	2092	Hello, You are invited to complete a survey fo...	0.061	0.000	0.939	0.8225
1	1567	I want to know about your story, how you got t...	0.123	0.028	0.849	0.7717
2	1550	Have any of you requested accommodations throu...	0.134	0.038	0.829	0.7301
3	2785	Whenever I have a reaction or meltdown, he alw...	0.145	0.055	0.800	0.9309
4	1936	Throwaway because this is an embarrassing issu...	0.066	0.215	0.719	-0.8853

SentimentIntensityAnalyzer
: pos[긍정], neg[부정], neu[중립], compound[복합] 감정 점수 반환

1] 전처리에 필요한 리스트 생성

의학 용어[병명] 풀이
대문자 줄임말 처리

```
# (대문자) 줄임말, 병명 전처리 리스트
upperletter_list = {# disorders, treatments
                    'PTSD':'posttraumatic stress disorder', 'OCD':'obsessive compulsive disorder', 'CPTSD':'complex Posttraumatic stress d
                    'ADHD':'attention deficit hyperactivity disorder','CBT':'cognitive behavioural therapy','SSRI':'selective serotonin re
                    'EMDR':'eye movement desensitization and reprocessing', 'DV':'domestic violence', 'GAD':'generalized anxiety disorder'
                    # acronyms
                    'BF':'boyfriend', 'GF':'girlfriend', 'FWB':'friend with benefit','ASAP':'as soon as possible',
                    'DR':'doctor','FB':'facebook','DM':'direct message','PM':'private message','MIL':'my in law','BTW':'by the way',
                    'IM':'i am','EBT':'electronic benefit transfer', 'LVN':'licensed vocational nurse', 'MS':'middle school',
                    'NYC':'new york city', 'US':'united states', 'UK':'united kingdom', 'VA':'veterans administration','CA':'california',
                    'BTW':'between', 'cuz':'because', 'a.k.a':'also known as', 'b****h':'bitch'}
```

1) 전처리에 필요한 리스트 생성

일반적인 약어 처리

```
# 약어 리스트
contractions = {"'cause": 'because', "I'd": 'I would', "I'd've": 'I would have', "I'll": 'I will', "I'll've": 'I will have', "I'm": 'I am
                "ain't": 'is not', "aren't": 'are not', "can't": 'cannot', "could've": 'could have', "couldn't": 'could not',
                "didn't": 'did not', "doesn't": 'does not', "don't": 'do not', "hadn't": 'had not', "hasn't": 'has not', "haven't": 'have
                "he'd": 'he would', "he'll": 'he will', "he's": 'he is', "here's": 'here is',
                "how'd": 'how did', "how'd'y": 'how do you', "how'll": 'how will', "how's": 'how is',
                "i'd": 'i would', "i'd've": 'i would have', "i'll": 'i will', "i'll've": 'i will have', "i'm": 'i am', "i've": 'i have',
                "isn't": 'is not', "it'd": 'it would', "it'd've": 'it would have', "it'll": 'it will', "it'll've": 'it will have', "it's"
                "let's": 'let us', "ma'am": 'madam', "mayn't": 'may not', "might've": 'might have', "mightn't": 'might not', "mightn't've
                "must've": 'must have', "mustn't": 'must not', "mustn't've": 'must not have', "needn't": 'need not', "needn't've": 'need
                "o'clock": 'of the clock', "oughtn't": 'ought not', "oughtn't've": 'ought not have', "sha'n't": 'shall not', "shan't": 'sh
                "she'd": 'she would', "she'd've": 'she would have', "she'll": 'she will', "she'll've": 'she will have', "she's": 'she is',
                "should've": 'should have', "shouldn't": 'should not', "shouldn't've": 'should not have', "so's": 'so as', "so've": 'so h
```

1) 전처리에 필요한 리스트 생성

reddit에서 생성된 말 처리

```
# 불용어 추가 리스트
stopwords_additional = ['url', 'tldr','tl dr','nbsp','edit','edited','chapter','update','id','madover']
```

Ex. ´tldr´: too long didn´t read

2] 전처리에 필요한 함수 생성

병명, 줄임말 전처리 함수

```
def upperletter_acronyms(text_column):
  text_revised=[]
  for post in text_column:
    for word in list(upperletter_list.keys()):
      post = post.replace(word, upperletter_list[word])
    text_revised.append(post)
  return text_revised
```

약어 수정, 소유격 제거 함수

```
import re

def short_words(text_column):
    specials = ["’", "‘", "´", "`"]
    for s in specials:
        text_column = text_column.replace(s, "'")
    text_revised=[]
    for post in text_column:
        for word in list(contractions.keys()):
            post = post.replace(word, contractions[word])
        text_revised.append(post)
    return text_revised
```

영어 외 문자 제거 함수

```
def only_english(text):
    only_eng = re.sub("[^a-zA-Z]", " ", text)
    return only_eng
```

2] 전처리에 필요한 함수 생성

짧은 문자 제외 함수

```
def rm_short_words(text):
    shortwords = re.compile(r'\W*\b\w{1}\b')
    return shortwords.sub('', text)
```

표제어 추출 함수

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
lemmatizer = WordNetLemmatizer()

def lemmatize_it(sent):
  empty = []
  for word, tag in nltk.pos_tag(word_tokenize(sent)): # 토큰화한 후 품사태깅, (원래단어, 품사) 반환
      wntag = tag[0].lower() # 품사의 첫글자 소문자로 저장
      wntag = wntag if wntag in ['a', 'r', 'n', 'v'] else None # r: adverb, v: verb, n: noun
      if not wntag: # 동사, 명사, 부사가 아니면
          lemma = word
          empty.append(lemma) # 원단어를 리스트에 추가
      else: # 동사, 명사, 부사면
          lemma = lemmatizer.lemmatize(word, wntag) # 동사, 명사, 부사면
          empty.append(lemma) # 표제어 추출해서(lemmatization) 리스트에 추가
  return ' '.join(empty) # 전체 단어들 문자열로 재결합
```

3] 최종 전처리 함수 생성

```
def text_preprocessing(text):
    # 줄임말, 병명 수정
    text['text'] = upperletter_acronyms(text['text'])

    # 약어 수정 및 소유격 제거
    text['text'] = short_words(text['text'])

    # 단어 토큰화
    text['text'] = text['text'].apply(lambda x: text_to_word_sequence(x))
    text['text'] = [' '.join(map(str,l)) for l in text['text']]

    # 숫자, 특수기호 제거
    text['text'] = text['text'].map(lambda x: only_english(x))

    # 짧은 단어(1문자) 제거
    text['text'] = text['text'].map(lambda x: rm_short_words(x))

    # 철자 수정
    spell = Speller()
    text['text'] = text['text'].apply(lambda x:spell(x))

    # 표제어 추출
    text['text'] = text['text'].apply(lemmatize_it)

    # 불용어 제거
    stop = stopwords.words('english')
    stop.append(stopwords_additional)
    text['text'] = text['text'].apply(lambda x:[word for word in x.split() if word not in stop])
    text['text'] = [' '.join(map(str,l)) for l in text['text']]

    return text['text']
```

4] 전처리 적용 결과

	post_id	text	pos	neg	neu	compound
0	2092	hello invited complete survey mu psychology de...	0.061	0.000	0.939	0.8225
1	1567	want know story get struggle never realize wou...	0.123	0.028	0.849	0.7717
2	1550	request accommodation university would willing...	0.134	0.038	0.829	0.7301
3	2785	whenever reaction meltdown always try give shi...	0.145	0.055	0.800	0.9309
4	1936	throwaway embarrass issue although everyone kn...	0.066	0.215	0.719	-0.8853

03 Modelling - ML Models

[1] ‘text´ column

1] Logistic Regression

- 모델링에 앞서 **train, valid** 데이터세트 나누기

```
ntrain = train.shape[0]
```

```
# 검증을 위해 X_train, X_valid 데이터셋 나누기
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(data_all[:ntrain], train['label'], test_size=0.3, random_state=42, stratify=train[['label']])
```

1] Logistic Regression-CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('cnt_vect', CountVectorizer(ngram_range=(1,2))),
    ('lr_clf', LogisticRegression())])

params = {'cnt_vect__ngram_range': [(1,1), (1,2), (1,3)],
          'cnt_vect__max_df': [100, 300, 700],
          'cnt_vect__max_features': [300, 500, 1000, None],
          'lr_clf__C': [1, 5, 10]
}

grid_cv_pipe_cnt = GridSearchCV(pipeline, param_grid=params, cv=3, scoring='accuracy', verbose=0)
grid_cv_pipe_cnt.fit(X_train['text'], y_train)
print(grid_cv_pipe_cnt.best_params_, grid_cv_pipe_cnt.best_score_)
lr_pred = grid_cv_pipe_cnt.predict(X_valid['text'])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid, lr_pred)))
```

```
{'cnt_vect__max_df': 700, 'cnt_vect__max_features': None, 'cnt_vect__ngram_range': (1, 3), 'lr_clf__C': 1} 0.7043007249075247
예측 정확도는 0.7030
```

1] Logistic Regression-TfidfVectorizer

```
from sklearn.feature_e
from sklearn.pipeline
from sklearn.linear_mo
from sklearn.metrics i
from sklearn.model_sel

pipeline = Pipeline([
    ('cnt_vect', Count
    ('lr_clf', Logisti

params = {'cnt_vect__n
          'cnt_vect__m
          'cnt_vect__m
          'lr_clf__C':
}

grid_cv_pipe_cnt = Gri
grid_cv_pipe_cnt.fit(X
print(grid_cv_pipe_cnt
lr_pred = grid_cv_pipe
print('예측 정확도는 {
```

```
{'cnt_vect__max_df': 7
예측 정확도는 0.7030
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(ngram_range=(1,2))),
    ('lr_clf', LogisticRegression())])

params = {'tfidf_vect__ngram_range': [(1,1), (1,2), (1,3)],
          'tfidf_vect__max_df': [100, 300, 700],
          'tfidf_vect__max_features': [300, 500, 1000, None],
          'lr_clf__C': [1, 5, 10]
}

grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3, scoring='accuracy', verbose=0)
grid_cv_pipe.fit(X_train['text'], y_train)
print(grid_cv_pipe.best_params_, grid_cv_pipe.best_score_)
lr_pred = grid_cv_pipe.predict(X_valid['text'])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid, lr_pred)))
```

```
{'lr_clf__C': 1, 'tfidf_vect__max_df': 700, 'tfidf_vect__max_features': None, 'tfidf_vect__ngram_range': (1, 1)} 0.7115063553536407
예측 정확도는 0.7215
```

- CountVectorizer와 TfidfVectorizer 성능 비교 결과, TfidfVectorizer의 성능이 더 좋음
  → 이후 모델들에서는 TfidfVectorizer만 적용함

2] XGBoost

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

pipeline=Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english')),
    ('xgb_clf', XGBClassifier())
])

params = {'tfidf_vect__ngram_range':[(1,1),(1,2),(1,3)],
          'xgb_clf__n_estimator': [100, 300, 500],
          'xgb_clf__learning_rate': [0.05, 0.1, 0.2],
          'xgb_clf__max_depth': [3, 4, 5, 6]}


grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3, scoring='accuracy', verbose=0)
grid_cv_pipe.fit(X_train['text'], y_train)
print(grid_cv_pipe.best_params_, grid_cv_pipe.best_score_)
xgb_pred = grid_cv_pipe.predict(X_valid['text'])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid, xgb_pred)))
```

```
{'tfidf_vect__ngram_range': (1, 1), 'xgb_clf__learning_rate': 0.2, 'xgb_clf__max_depth': 5, 'xgb_clf__n_estimator': 100} 0.6747970507187011
예측 정확도는 0.6862
```

3] RandomForest

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(ngram_range=(1,2))),
    ('rf_clf', RandomForestClassifier())])

params = {'tfidf_vect__ngram_range': [(1,1), (1,2), (1,3)],
          'tfidf_vect__max_df': [100, 300, 700],
          'rf_clf__n_estimators': [5, 50, 100],
          'rf_clf__max_depth': [2, 10, 20],
}

grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3, scoring='accuracy', verbose=0)
grid_cv_pipe.fit(X_train['text'], y_train)
print(grid_cv_pipe.best_params_, grid_cv_pipe.best_score_)
rf_pred = grid_cv_pipe.predict(X_valid['text'])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid, rf_pred)))
```

```
{'rf_clf__max_depth': 20, 'rf_clf__n_estimators': 100, 'tfidf_vect__max_df': 300, 'tfidf_vect__ngram_range': (1, 2)} 0.6870407263970607
예측 정확도는 0.6711
```

4] Support Vector Machine[SVM]

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(ngram_range=(1,2))),
    ('svm_clf', SVC())])

params = {'tfidf_vect__ngram_range': [(1,1), (1,2), (1,3)],
          'tfidf_vect__max_df': [100, 300, 700],
          'svm_clf__C': [0.01, 0.1, 1, 5, 10],
          'svm_clf__gamma': [0.01, 0.1, 1, 5, 10] }

grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3, scoring='accuracy', verbose=0)
grid_cv_pipe.fit(X_train['text'], y_train)
print(grid_cv_pipe.best_params_, grid_cv_pipe.best_score_)
svm_pred = grid_cv_pipe.predict(X_valid['text'])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid, svm_pred)))
```

```
{'svm_clf__C': 1, 'svm_clf__gamma': 1, 'tfidf_vect__max_df': 300, 'tfidf_vect__ngram_range': (1, 1)} 0.706471413321417
예측 정확도는 0.7181
```

03 Modelling - ML Models

[2] ‘text´ column
+´sentiment´ column

```
# TF-IDF tVectorizer를 이용한 피처 벡터화 함수
def tf_vector(text):
  vect = TfidfVectorizer()
  text_vector = pd.DataFrame(vect.fit_transform(text).toarray())
  text_vector = text_vector.add_prefix("tf_")
  return text_vector

data_all_tf_vect = pd.concat([data_all, tf_vector(data_all['text'])], axis=1)
data_all_tf_vect.head()
```

	post_id	text	pos	neg	neu	compound	tf_0	tf_1	tf_2	tf_3	...	tf_7927	tf_7928	tf_7929	tf_7930	tf_7931
0	2092	hello invite complete survey mu psychology dep...	0.061	0.000	0.939	0.8225	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
1	1567	want know story get struggle never realize wou...	0.123	0.028	0.849	0.7717	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
2	1550	request accommodation university would willing...	0.134	0.038	0.829	0.7301	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

```
# 검증을 위해 X_train, X_valid 데이터셋 나누기
from sklearn.model_selection import train_test_split
X_train_sent, X_valid_sent, y_train_sent, y_valid_sent = train_test_split(data_all_tf_vect[:ntrain], train['label'], test_size=0.3, random_state=42, stratify=train[['label']])
```

1] Logistic Regression

```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lr = LogisticRegression()
lr.fit(X_train_sent.iloc[:, 2:], y_train_sent)
lr_sent_pred = lr.predict(X_valid_sent.iloc[:,2:])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid_sent, lr_sent_pred)))
```

```
예측 정확도는 0.7198
```

2] XGBoost

```
# Xgboost
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train_sent.iloc[:, 2:], y_train_sent)
xgb_sent_pred = xgb.predict(X_valid_sent.iloc[:,2:])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid_sent, xgb_sent_pred)))
```

```
예측 정확도는 0.7215
```

3] RandomForest

- 성능이 가장 좋은 RandomForest 모델만 파라미터 튜닝

```
# RandomForest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=300, max_features='sqrt', min_samples_split=4)
rf.fit(X_train_sent.iloc[:,2:], y_train_sent)
rf_sent_pred = rf.predict(X_valid_sent.iloc[:,2:])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid_sent, rf_sent_pred)))
```

```
예측 정확도는 0.7349
```

4] SVM

```
# SVM
import sklearn.svm as svm
svm_clf = svm.SVC(kernel='rbf')
svm_clf.fit(X_train_sent.iloc[:,2:], y_train_sent)
svm_sent_pred = svm_clf.predict(X_valid_sent.iloc[:,2:])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid_sent, svm_sent_pred)))
```

```
예측 정확도는 0.7047
```

3] RandomForest

```
params = {
    'n_estimators': [150, 250, 350],
    'max_features': ['sqrt', 0.25, 0.5, 0.75],
    'min_samples_split': [2, 4, 6]
}

grid_rf = GridSearchCV(rf, params, cv=3)
grid_rf.fit(X_train_sent.iloc[:,2:], y_train)
print(grid_rf.best_estimator_)
print(grid_rf.best_score_)

grid_rf_pred = grid_rf.predict(X_valid_sent.iloc[:,2:])
print('예측 정확도는 {0:.4f}'.format(accuracy_score(y_valid_sent, grid_rf_pred)))
```

```
RandomForestClassifier(max_features='sqrt', min_samples_split=4,
                       n_estimators=250)
0.7431574439562075
```

04 Modelling - DL Models

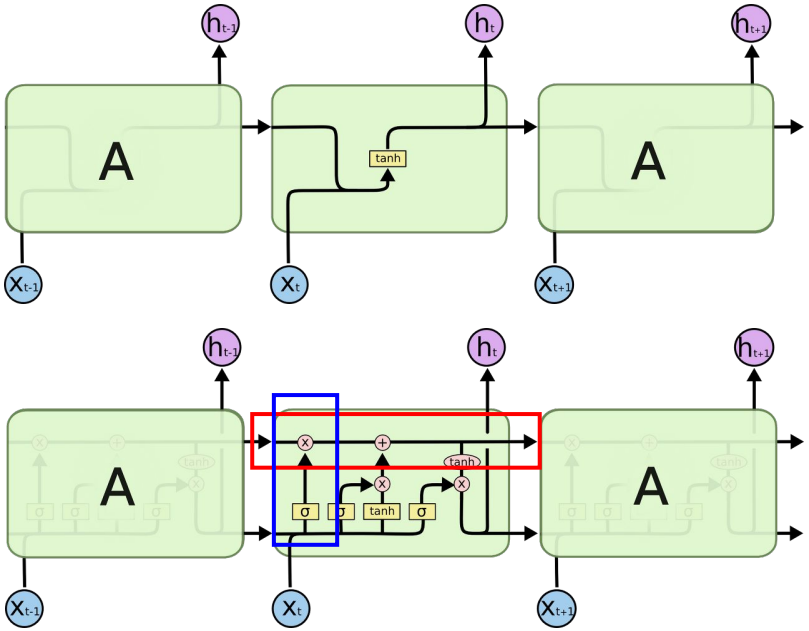
(1) LSTM

RNN (Recurrent neural network)

- 이전에 일어난 사건을 바탕으로 나중에 일어나는 사건을 생각하지 못하는 전통적인 neural network의 문제를 해결
- 스스로를 반복하면서 **이전 단계에서 얻은 정보가 지속**되도록 함
- But 필요한 정보를 얻기 위한 시간 격차가 굉장히 커짐
- 긴 기간의 의존성[long-term dependencies]

LSTM (Long Short-Term Memory)

- RNN의 특별한 한 종류로, 긴 의존 기간을 필요로 하는 학습을 수행할 능력
- cell state에 뭔가를 더하거나 없앨 수 있는 기능이 gate라고 불리는 구조에 의해서 조심스럽게 제어
- Sigmoid layer는 0과 1 사이의 숫자를 내보내는데, 그 값이 0이라면 아무 것도 넘기지 않고, 1이라면 모든 것을 넘김
- LSTM은 **3개의 gate**를 가지고 있고, 이 문들은 cell state를 보호하고 제어



LSTM용 전처리

- 텍스트 토큰화 후 데이터에서 등장하는 단어의 빈도수 count
- 1번만 등장하는 희귀단어의 비율과 총 토큰 대비 비율 확인

빈도수가 2 이하인 단어를 제거하도록 vocab_size 설정

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train['text'])
```

```
threshold = 2
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)*100)
```

```
단어 집합(vocabulary)의 크기 : 4343
등장 빈도가 1번 이하인 희귀 단어의 수: 2002
단어 집합에서 희귀 단어의 비율: 46.09716785632052
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 6.906067818827831
```

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

```
단어 집합의 크기 : 2342
```

LSTM용 전처리

- 설정한 단어사전 크기에 맞춰 토큰화 텍스트를 시퀀스 형태로 변환
- 각 텍스트별 단어 개수 분포 시각화 [최대 길이 119, 평균 길이 35]

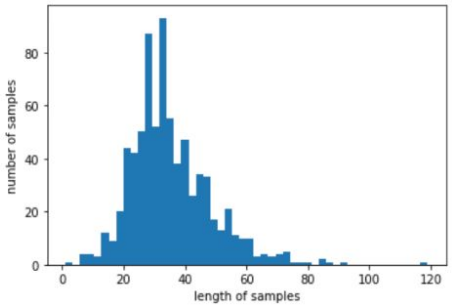
최대 단어 길이 [maxlen] 100으로 설정해 패딩 진행

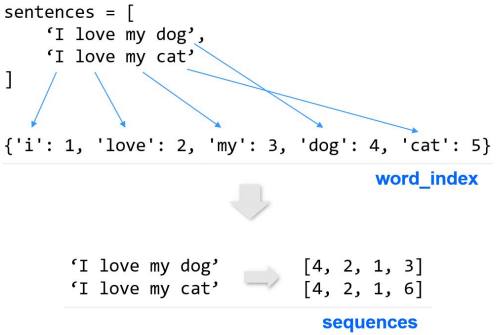
```
tokenizer = Tokenizer(vocab_size)
tokenizer.fit_on_texts(X_train['text'])
```

```
X_train['text'] = tokenizer.texts_to_sequences(X_train['text'])
X_valid['text'] = tokenizer.texts_to_sequences(X_valid['text'])

y_train = np.array(y_train)
y_valid = np.array(y_valid)
```

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

```
텍스트의 최대 길이 : 119
텍스트의 평균 길이 : 35.36959370904325
```





* 시퀀스 형태란?

LSTM용 전처리

- 설정한 단어사전 크기에 맞춰 토큰화 텍스트를 시퀀스 형태로 변환
- 각 텍스트별 단어 개수 분포 시각화 [최대 길이 119, 평균 길이 35]

최대 단어 길이 [maxlen] 100으로 설정해 패딩 진행

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

```
max_len = 100
below_threshold_len(max_len, X_train['text'])
```

```
전체 샘플 중 길이가 100 이하인 샘플의 비율: 99.8689384010485
```

```
from keras_preprocessing.sequence import pad_sequences
X_train_padded = pad_sequences(X_train['text'], maxlen=max_len)
X_valid_padded = pad_sequences(X_valid['text'], maxlen=max_len)
```

* 패딩[padding] : 데이터에 특정값을 채워서 데이터의 크기[shape]를 조정하는 것

```
] 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

  embedding_dim = 100
  hidden_units = 128

  model = Sequential()
  model.add(Embedding(vocab_size, embedding_dim))
  model.add(LSTM(hidden_units))
  model.add(Dense(1, activation='sigmoid'))

  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(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  history = model.fit(X_train_padded, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

```
Epoch 1/15
10/10 [==============================] - ETA: 0s - loss: 0.6916 - acc: 0.5361
Epoch 1: val_acc improved from -inf to 0.50980, saving model to best_model.h5
10/10 [==============================] - 6s 286ms/step - loss: 0.6916 - acc: 0.5361 - val_loss: 0.6856 - val_acc: 0.5098
Epoch 2/15
10/10 [==============================] - ETA: 0s - loss: 0.6698 - acc: 0.6148
```

I	:	0.2	0.58	0.72
am	:	0.4	0.3	0.39
a	:	0.5	0.82	0.22

* 임베딩[embedding] : 단어를 벡터로 표현하되, 희소 벡터가 아닌 밀집 벡터로 변환

01 딥러닝 모델 구축

- 임베딩 레이어
  : 단어사전 크기와 임베딩 차원을 파라미터로 설정
- LSTM 레이어
  : 은닉유닛 설정

02 모델 컴파일

- earlystopping 설정
- checkpoint로 학습모델 저장

03 모델 훈련

- 패딩된 훈련셋 입력

04 모델 평가

- 검증셋 정확도 0.6881

```
] 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

  embedding_dim = 100
  hidden_units = 128

  model = Sequential()
  model.add(Embedding(vocab_size, embedding_dim))
  model.add(LSTM(hidden_units))
  model.add(Dense(1, activation='sigmoid'))

  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(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  history = model.fit(X_train_padded, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

```
Epoch 1/15
10/10 [==============================] - ETA: 0s - loss: 0.6916 - acc: 0.5361
Epoch 1: val_acc improved from -inf to 0.50980, saving model to best_model.h5
10/10 [==============================] - 6s 286ms/step - loss: 0.6916 - acc: 0.5361 - val_loss: 0.6856 - val_acc: 0.5098
Epoch 2/15
10/10 [==============================] - ETA: 0s - loss: 0.6698 - acc: 0.6148
```

I	:	0.2	0.58	0.72
am	:	0.4	0.3	0.39
a	:	0.5	0.82	0.22

* 임베딩[embedding] : 단어를 벡터로 표현하되, 희소 벡터가 아닌 밀집 벡터로 변환

01 딥러닝 모델 구축

- 임베딩 레이어
  : 단어사전 크기와 임베딩 차원을 파라미터로 설정
- LSTM 레이어
  : 은닉유닛 설정

02 모델 컴파일

- earlystopping 설정
- checkpoint로 학습모델 저장

03 모델 훈련

- 패딩된 훈련셋 입력

04 모델 평가

- 검증셋 정확도 0.6881

```
] 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

  embedding_dim = 100
  hidden_units = 128

  model = Sequential()
  model.add(Embedding(vocab_size, embedding_dim))
  model.add(LSTM(hidden_units))
  model.add(Dense(1, activation='sigmoid'))

  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(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  history = model.fit(X_train_padded, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

```
Epoch 1/15
10/10 [==============================] - ETA: 0s - loss: 0.6916 - acc: 0.5361
Epoch 1: val_acc improved from -inf to 0.50980, saving model to best_model.h5
10/10 [==============================] - 6s 286ms/step - loss: 0.6916 - acc: 0.5361 - val_loss: 0.6856 - val_acc: 0.5098
Epoch 2/15
10/10 [==============================] - ETA: 0s - loss: 0.6698 - acc: 0.6148
```

I	:	0.2	0.58	0.72
am	:	0.4	0.3	0.39
a	:	0.5	0.82	0.22

* 임베딩[embedding] : 단어를 벡터로 표현하되, 희소 벡터가 아닌 밀집 벡터로 변환

01 딥러닝 모델 구축

- 임베딩 레이어
  : 단어사전 크기와 임베딩 차원을 파라미터로 설정
- LSTM 레이어
  : 은닉유닛 설정

02 모델 컴파일

- earlystopping 설정
- checkpoint로 학습모델 저장

03 모델 훈련

- 패딩된 훈련셋 입력

04 모델 평가

- 검증셋 정확도 0.6881

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

```
11/11 [==============================] - 2s 89ms/step - loss: 0.5659 - acc: 0.6881

 테스트 정확도: 0.6881
```

01 딥러닝 모델 구축	02 모델 컴파일	03 모델 훈련	04 모델 평가
- 임베딩 레이어 : 단어사전 크기와 임베딩 차원을 파라미터로 설정 - LSTM 레이어 : 은닉유닛 설정	- earlystopping 설정 - checkpoint로 학습모델 저장	- 패딩된 훈련셋 입력	- 검증셋 정확도 0.6881

파라미터 조절해 2차 모델링

maxlen

- 패딩 관련 파라미터
- 최대 단어길이를 포함하도록 증가시킴

output_dim

- 임베딩 관련 파라미터
- vocab_size 증가시킴

```
vocab_size = 3000
max_len = 120

X_train_padded = pad_sequences(X_train['text'], maxlen=max_len)
X_valid_padded = pad_sequences(X_valid['text'], maxlen=max_len)
```

```
embedding_dim = 100
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))

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

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train_padded, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

```
Epoch 1/15
10/10 [==============================] - ETA: 0s - loss: 0.6924 - acc: 0.5295
Epoch 1: val_acc improved from -inf to 0.51634, saving model to GRU_model.h5
10/10 [==============================] - 5s 280ms/step - loss: 0.6924 - acc: 0.5295 - val_loss: 0.6906 - val_acc: 0.5163
Epoch 2/15
10/10 [==============================] - ETA: 0s - loss: 0.6236 - acc: 0.6508
Epoch 2: val_acc did not improve from 0.51634
10/10 [==============================] - 2s 228ms/step - loss: 0.6236 - acc: 0.6508 - val_loss: 5.5700 - val_acc: 0.5033
Epoch 3/15
10/10 [==============================] - ETA: 0s - loss: 1.4359 - acc: 0.7541
Epoch 3: val_acc improved from 0.51634 to 0.64052, saving model to GRU_model.h5
```

파라미터 조절해 2차 모델링

maxlen

- 패딩 관련 파라미터
- 최대 단어길이를 포함하도록 증가시킴

output_dim

- 임베딩 관련 파라미터
- vocab_size 증가시킴

검증셋 정확도
0. 7584

LSTM_.csv
Complete · sehan25 · 3d ago

0.68288

```
vocab_size = 3000
max_len = 120

X_train_padded = pad_sequences(X_train['text'], maxlen=max_len)
X_valid_padded = pad_sequences(X_valid['text'], maxlen=max_len)
```

```
embedding_dim = 100
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))

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

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train_padded, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

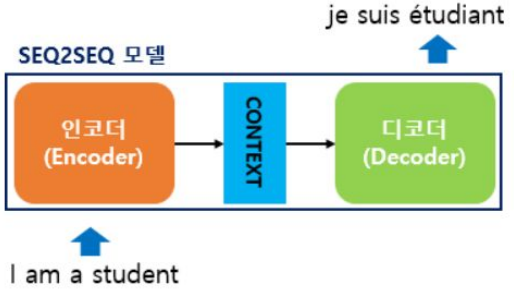
```
Epoch 1/15
10/10 [==============================] - ETA: 0s - loss: 0.6924 - acc: 0.5295
Epoch 1: val_acc improved from -inf to 0.51634, saving model to GRU_model.h5
10/10 [==============================] - 5s 280ms/step - loss: 0.6924 - acc: 0.5295 - val_loss: 0.6906 - val_acc: 0.5163
Epoch 2/15
10/10 [==============================] - ETA: 0s - loss: 0.6236 - acc: 0.6508
Epoch 2: val_acc did not improve from 0.51634
10/10 [==============================] - 2s 228ms/step - loss: 0.6236 - acc: 0.6508 - val_loss: 5.5700 - val_acc: 0.5033
Epoch 3/15
10/10 [==============================] - ETA: 0s - loss: 1.4359 - acc: 0.7541
Epoch 3: val_acc improved from 0.51634 to 0.64052, saving model to GRU_model.h5
```

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

```
11/11 [==============================] - 1s 26ms/step - loss: 0.5698 - acc: 0.7584

테스트 정확도: 0.7584
```

Seq2seq 모델

- 인코더-디코더 구조로 구성되어 있음.
  인코더: 입력 시퀀스를 하나의 벡터 표현으로 압축
  디코더: 이 벡터 표현을 통해서 출력 시퀀스를 만들어냄.



Seq2seq 모델의 한계점

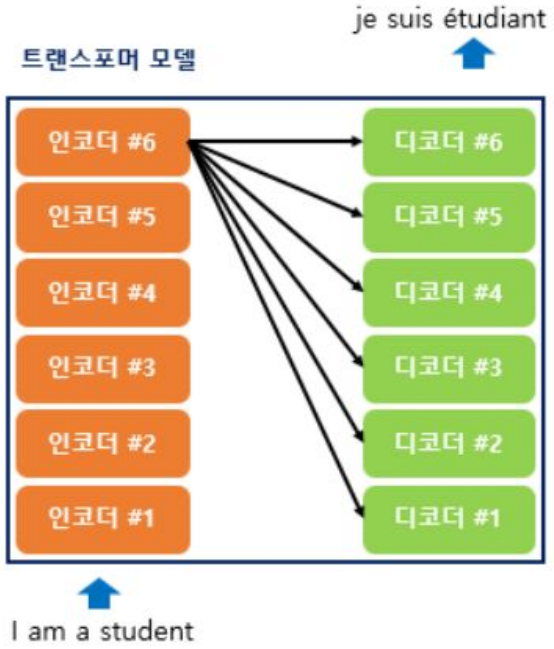
- 인코더가 입력 시퀀스를 하나의 벡터로 압축하는 과정에서 길이가 긴 문장의 경우 입력 시퀀스의 정보가 일부 손실됨.
- Input에 입력되는 순서대로(sequential) 문장을 처리하여 연산 속도가 매우 느림.

Transformer 모델

- 기존의 seq2seq의 구조인 인코더-디코더를 따르면서 어텐션[Attention]만으로 구현한 모델

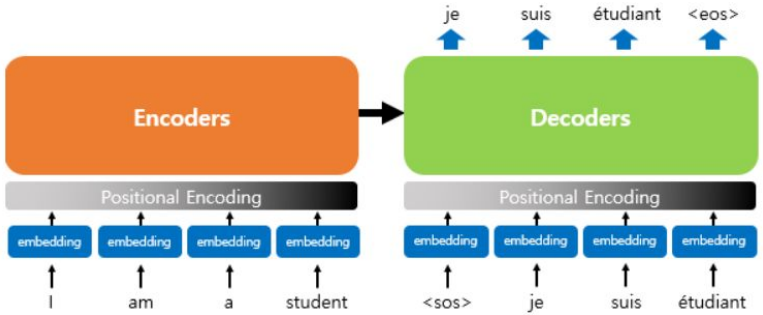
- 인코더와 디코더라는 단위가 n개로 구성되는 구조

- 입력한 문장을 병렬로 한번에 처리하여 연산속도가 매우 빠름.



Transformer 모델

- 단어 입력을 병렬로 처리하므로 단어의 위치 정보를 알려주기 위해 Positional Encoding을 함.
- Positional Encoding: 각 단어의 임베딩 벡터에 위치 정보들을 더하여 모델의 입력으로 사용함.



simple transformers 모델

* RoBERTa model: BERT 모델의 파생 모델로, 기존 BERT 모델에 추가적인 학습 방법을 제시하여 성능을 향상시킨 모델

```
from simpletransformers.classification import ClassificationModel

# Create a TransformerModel
model = ClassificationModel('roberta', 'roberta-base', num_labels=2, args={'reprocess_input_data': True, 'overwrite_output_dir': True},use_cuda=False)

# Train
model.train_model(train_df)

# Evaluate with validation data
result, model_outputs, wrong_predictions = model.eval_model(valid_df)
```

모델 생성: ['bert', 'xlnet', 'xlm', 'roberta', 'distilbert']

Train

Test

result

```
{'mcc': 0.6908666906154659,
 'tp': 280,
 'tn': 224,
 'fp': 57,
 'fn': 35,
 'auroc': 0.9256171270406146,
 'auprc': 0.9333060103034989,
 'eval_loss': 0.35800740743676823}
```

model_outputs

레이블 별 모델 결과 -> 값이 클수록 해당 레이블일 확률이 더 높음

```
array([[-1.67968035,  1.81450522],
       [-1.64466035,  1.66866684],
       [-1.71534121,  1.6734066 ],
       ...,
       [ 1.87525845, -2.17779922],
       [ 0.0941721 , -0.35580754],
       [ 0.93230498, -0.99792325]])
```

```
lst = []
for arr in model_outputs:
    lst.append(np.argmax(arr))
```

모델 결과가 더 큰 레이블 값 배치

```
true = valid_df['label'].tolist()
predicted = lst
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
mat = confusion_matrix(true , predicted)
print(mat)
print(accuracy_score(true, predicted))
```

```
[[224  57]
 [ 35 280]]
0.8456375838926175
```

validation data로 측정한 confusion matrix

validation data로 측정한 accuracy

```
model.train_model(train[['text','label']])
```

Train 데이터와 validation 데이터를 합쳐 재학습

```
def get_result(statement):
    result = model.predict([statement])
    pos = np.where(result[1][0] == np.amax(result[1][0]))
    pos = int(pos[0])
    return pos
```

Test 데이터로 예측

```
predict = pd.DataFrame()
predict['label'] = X_test['text'].apply(lambda x: get_result(x))
```

```
X_test['label'] = predict['label']
```

```
X_test[['post_id','label']].to_csv('roberta_전처리x.csv', index=False)
```

bert.csv Complete · Jihaeraguyo · 5d ago	전처리 진행한 데이터	0.75671
bert_x.csv Complete · Jihaeraguyo · 2d ago	전처리하지 않은 데이터	0.77181
roberta_x.csv Complete · Jihaeraguyo · 18h ago	전처리하지 않은 데이터	0.84563

#	Team	Members	Score	Entries	Last	Join
1	**NoStress**		0.84563	10	18h	
	Your Best Entry! Your most recent submission scored 0.84563, which is an improvement of your previous score of 0.77181. Great job!				Tweet this	
2	Aya Mourad		0.82550	2	20d	
3	Assanali Abu		0.82214	3	8d	
4	GEORGE DOUKAS		0.82046	2	23d	
5	Phan Minh Toan		0.81879	3	9d	
6	kaled hoshme		0.80704	4	9d	
7	JK		0.80033	1	22d	
8	Yakov Kononov		0.77181	2	23d	

05 Conclusion

Classical ML Algorithms

(1) 데이터 전처리 및 가공을 통해서 문제 해결에 적합한 **특징들을 추출**
(2) 이를 학습 데이터로 하여 **패턴 분류기를 훈련**시키는
두 개의 단계로 문제 해결

전처리 및 feature engineering 을 통해 손실되는 정보 발생 가능

DL Algorithms

- 특징 추출을 위한 전처리 단계를 전체 학습 프로세스에 포함시킴으로써 가공되지 않은 원래 데이터를 직접 학습하도록 하는 **통합된 방식의 문제해결**
- 차원수가 아주 크고 복잡한 데이터의 경우에전처리 과정을 통해서 손실될 수도 있는 정보를 기계가 자동으로 추출해서 활용

기존의 전처리 방법이나 소위 feature engineering 을 통해 배재되었던 해의 영역 조차도 탐색해 유용한 정보 추출

01 학기 세션중 배운 텍스트마이닝 및 딥러닝 모델 활용

02 딥러닝 모델의 파라미터 설정 및 레이어 설계 방식을 구체적으로 알지 못해 외부 노트북을 많이 참고

03 딥러닝 성능을 올릴 수 있는 튜닝 혹은 전처리 방식에 대해 앞으로 더 공부하고자 함

Stress Detection
on Social Media Posts

OB TEAM 2