AI야, 진짜뉴스를 찾아줘!

★(신)호둥이★ (이서희 이찬영 윤승후)

목차

1. 사용한 라이브러리

a) fastText 설치

```
!wget https://github.com/facebookresearch/fastText/archive/v0.9.2.zip
!unzip v0.9.2.zip
%cd fastText-0.9.2
!make
```

b) Soynlp

문서 단위 말뭉치 생성 기능

```
from soynlp.utils import DoublespaceLineCorpus
from soynlp.noun import LRNounExtractor_v2
```

명사 추출 라이브러리

c) Mecab

Mecab으로 토큰화 진행

```
from konlpy.tag import Mecab
from jamo import h2j, j2hcj
```

종성 여부 판단 기능

d) 기타

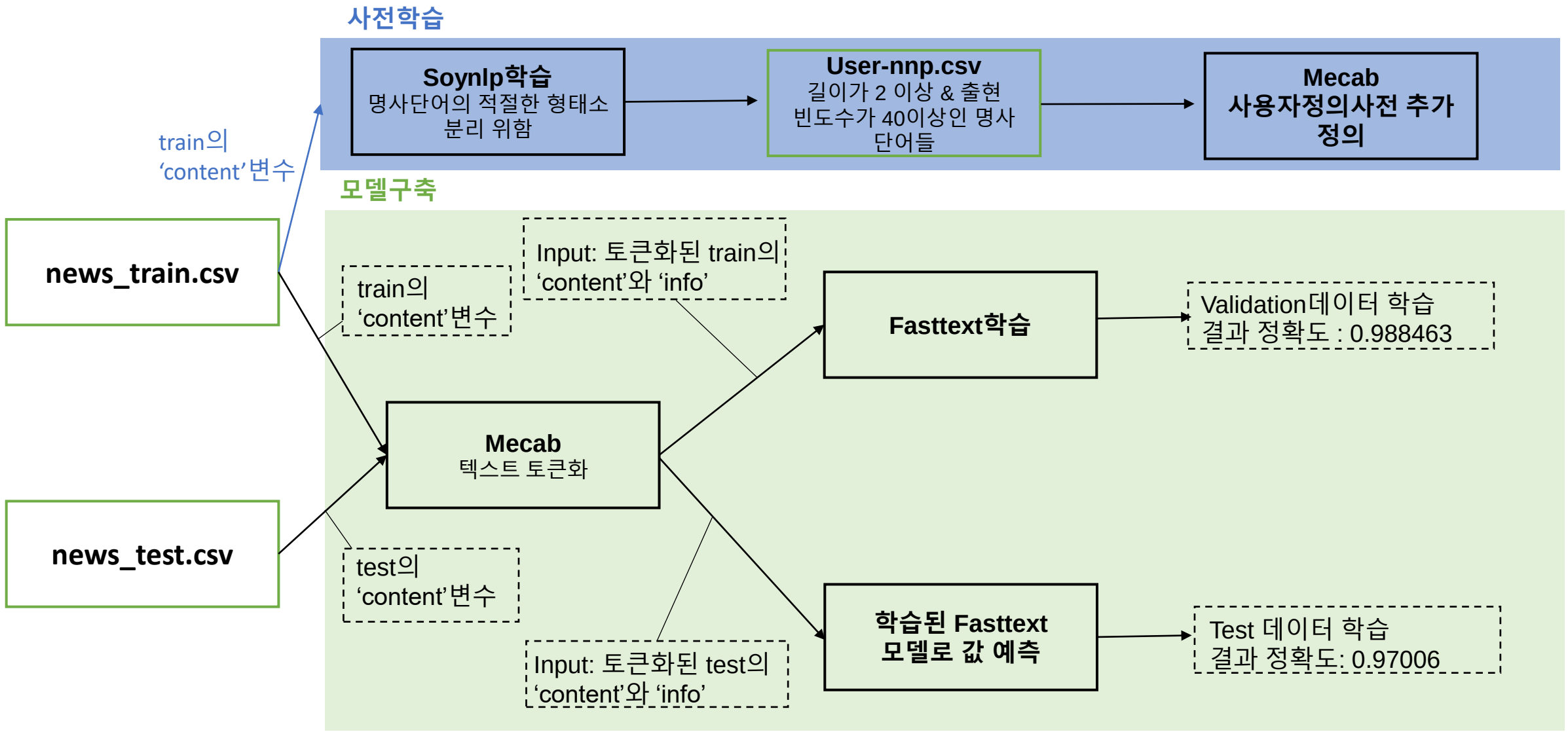
Pandas 라이브러리

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
```

데이터 분할 라이브러리 (train, validation)

2. 분석 프로세스


사전학습
Soynlp학습
명사단어의 적절한 형태소 분리 위함
User-nnp.csv
길이가 2 이상 & 출현 빈도수가 40이상인 명사 단어들
Mecab
사용자정의사전 추가 정의
train의 'content'변수
모델구축
news_train.csv
train의 'content'변수
Input: 토큰화된 train의 'content'와 'info'
Fasttext학습
Validation데이터 학습 결과 정확도 : 0.988463
Mecab
텍스트 토큰화
news_test.csv
test의 'content'변수
학습된 Fasttext 모델로 값 예측
Test 데이터 학습 결과 정확도: 0.97006
Input: 토큰화된 test의 'content'와 'info'

3) Soynlp

3-1) Soynlp 사용배경 및 필요성

- **배경:**

- mecab을 이용하여 텍스트의 형태소를 분리하는 과정 중, **신조어나 형태소 분석기에 등록되지 않은 단어의 경우 형태소 분리가 제대로 이루어지지 않음**을 발견

예) '하이스탁론'-> '하이스', '탁론' / '이데일리' -> '이', '데일리 '

```
print(mecab.pos('하이스탁론, 선취수수료 없는'))
print(mecab.pos('종합 경제정보 미디어 이데일리'))
```

```
[('하이스', 'NNP'), ('탁론', 'NNG'), (',', 'SC'), ('선취', 'NNG'), ('수수료', 'NNG'), ('없', 'VA'), ('는', 'ETM')]
[('종합', 'NNG'), ('경제', 'NNG'), ('정보', 'NNG'), ('미디어', 'NNG'), ('이', 'MM'), ('데일리', 'NNP')]
```

- **필요성:**

- soynlp는 텍스트 데이터에서 함께 자주 등장하는 특정 문자 시퀀스가 앞 뒤로 다양한 단어들을 동반한다면, 해당 문자 시퀀스를 하나의 독립된 단어로 파악
- soynlp를 사용할 시, **mecab으로 제대로 분리되지 않았던** '하이스탁론 ' 이나 '이데일리'와 같이 기존에 한 단어로 인식하지 못했던 단어들을 제대로 인식 예상
- 신조어가 많이 발생하고 단어의 수가 많은 **명사의 적절한 분리가 모델의 성능향상에 중요한 요소가 될 것이라 판단**

- **사용된 모듈**:

- **LRNounExtractor_v2**: A단어 다음에 등장하는 단어들에 대해서 A의 명사가능점수를 미리 계산해 놓은 수치를 활용하여, A의 명사점수를 계산. 계산된 수치를 바탕으로 **A가 하나의 명사인지 아닌지 판단**

3) SoynIp

3-2) soynlp의 LRNounExtractor_v2 작동원리

```
noun_extractor.lrgraph.get_r('하이스탁론')

[(',', 1078), ('', 17), ('에서', 5), ('의', 4)]
```

→ '하이스탁론' 뒤에 붙을 수 있는 단어들 출력

a) 세종말뭉치로부터 학습된 각 R parts(',', '', '에서', '의')에 대한 r_scores가져오기

→

b) ' 하이스탁론'에 대한 명사 점수 계산
{(1078 * ','에 대한 r_score) + (17 * ''에 대한 r_score) + (5 * '에서'에 대한 r_score) + (4 * '의'에 대한 r_score)} / (1078 + 17 + 5 + 4)

→

c) '하이스탁론' 의 발생 빈도수와 명사점수 출력

```
nouns = noun_extractor.train_extract(text1)
nouns['하이스탁론']

NounScore(frequency=26, score=1.0)
```

a) r_scores 학습:

- 한국어가 지니는 L+[R]구조(의미를 가지는 단어로 명사, 동사 등 + [문법기능을 하는 조사])로 인해, **R을 통해 L이 어떤 품사를 지니는지 유추** 가능
- Soynlp에선 품사라벨이 붙어있는 **국립국어원의 세종말뭉치 데이터를 학습 데이터로 사용**. 세종말뭉치의 단어들을 L+R로 분해한 후, R parts를 features로 가지는 Logistic regression을 활용하여 L parts가 명사인지 아닌지 판단. 해당 과정에서 **산출된 coefficients에 후처리 적용 후 R parts에 대한 r_scores로 이용 -> 해당 회귀 분석은 soynlp에서 사전 학습되어 알고리즘에 사용 중인 모델**

b) L단어의 명사 점수 계산:

- **L+[R]의 발생 빈도수에 a에서 계산된 r_scores를 각각 곱한 값의 합계를 발생 빈도수의 합계로 나눔**
- 명사 점수가 특정 threshold 이상일 때, 해당 L단어를 명사로 추출

c) L단어가 등장한 빈도수와 명사점수 출력:

- L단어가 **실제로 명사로 쓰였다고 추정되었을 때의 발생 빈도 수와 명사점수 출력** 가능

3) SoynIp

3-3) soynlp 실행과정

a) Train 데이터 셋의 문장들을 리스트로 받아 text1에 저장

```
text1
```

```
['[이데일리 MARKETPOINT]15:32 현재 코스닥 기관 678억 순매도',
 ' 실적기반  저가에 매집해야 할 8월 급등유망주 TOP 5 전격공개',
 '하이스탁론, 선취수수료 없는 월 0 4% 최저금리 상품 출시',
```

b) LRNounExtractor_v2를 통해 text1 학습시키고 noun의 빈도수와 scores를 nouns에 저장

```
noun_extractor = LRNounExtractor_v2(verbose=True)
nouns = noun_extractor.train_extract(text1)
```

```
[Noun Extractor] use default predictors
```

c) Nouns에서 40번 이상 발생하고 scores가 0.5이상인 단어들 중, 단어 길이가 2이상인 단어들만 tgt_noun에 추가

```
tgt_noun = []
for noun in nouns:
    if (nouns[noun][0] >= 40) and (nouns[noun][1]) >= 0.5:
        if len(noun) >= 2:
            tgt_noun.append(noun)
```

d) tgt_noun들의 단어 중 '받아들 ' , '기준) ' 과 같이 잘 정제되지 않은 단어들이 존재하여, 해당 단어들에서 제거되어야 할 단어들을 골라내어 추가 정제

```
non_noun_eomi_list = ('.', ',', '(',')','들','지')
tgt_noun = set([item.strip("""',. ' "[]()" '  """) for item in tgt_noun if (item[-1] not in non_noun_eomi_list) #
                & (sum([1 if char.isdigit() else 0 for char in item]) <= 2) #
                & (len(re.sub('[^가-힣]', '', item)) <= 2) #
                & (not re.search('[^가-힣0-9]', item))])
```

e) 추가 정제된 단어들을 user_nnp.csv파일로 저장

4. Mecab 사용자 정의사전 구축 프로세스

a) 종성 여부 판단하는 함수 만들기

```
def get_jongsung_TF(sample_text):
   sample_text_list = list(sample_text)
   last_word = sample_text_list[-1]
   last_word_jamo_list = list(j2hcj(h2j(last_word)))
   last_jamo = last_word_jamo_list[-1]
   jongsung_TF = "T"

   if last_jamo in ['ㅏ', 'ㅑ', 'ㅓ', 'ㅕ', 'ㅗ', 'ㅛ', 'ㅜ', 'ㅠ', 'ㅡ', 'ㅣ', 'ㅘ', 'ㅚ', 'ㅐ', 'ㅝ', 'ㅖ', 'ㅢ', 'ㅒ,ㅖ', 'ㅟ', 'ㅔ',
     jongsung_TF = "F"
   return jongsung_TF
```

- 한글은 종성여부에 따라 뒤의 조사가 결정되기 때문에 앞글자(명사단어)가 종성으로 끝날때와 종성으로 끝나지 않을 때의 조사가 다름
- 해당 단어에 적합한 조사규칙을 적용하기 위해 종성여부를 판단하는 함수 필요
- 사용자 정의 사전 특정 칼럼에 인식을 할 수 있도록 종성 여부 함수 정의

b) SoynIp 파일 불러오기

```
mecab_dict = pd.read_csv('/content/mnt/Shareddrives/Real_News/찬영/noun.csv',encoding='cp949')
```

4. Mecab 사용자 정의사전 구축 프로세스

c) Mecab 사용자 정의 사전 불러오기

```
with open("./user-dic/nnp.csv", 'r', encoding='utf-8') as f:
  file_data = f.readlines()
```

d) 사용자 정의 사전 추가

```
for word in word_list:
  jongsung_TF = get_jongsung_TF(word)
  if len(word)>=4:
    line = '{},,,0,NNP,*,{},{},*,*,*,*,*₩n'.format(word , jongsung_TF, word)
    file_data.append(line)

  elif len(word) == 3:
    line = '{},,,1,NNP,*,{},{},*,*,*,*,*₩n'.format(word , jongsung_TF, word)
    file_data.append(line)

  elif len(word) == 2:
    line = '{},,,2,NNP,*,{},{},*,*,*,*,*₩n'.format(word , jongsung_TF, word)
    file_data.append(line)
```

단어 길이에 따라 우선순위를 0,1,2로 지정

- word_list는 soynlp 데이터의 리스트 형태
- 길이가 긴 단어들을 우선적으로 토큰화
- nnp.csv 파일에 저장

e) user_nnp 파일에 업로드

```
!bash ./tools/add-userdic.sh
```

명사 컬럼 우선순위 컬럼

	대우	1786	3545	3821	NNP	*	F	대우.1	*.1	*.2	*.3	*.4	*.5
0	구글	1786	3546	2953	NNP	*	T	구글	*	*	*	*	*
1	에이프로젠	1786	3546	2953	NNP	*	T	에이프로젠	*	*	*	*	*
2	의료현장	1786	3546	2953	NNP	*	T	의료현장	*	*	*	*	*
3	목표주가	1786	3545	2953	NNP	*	F	목표주가	*	*	*	*	*
4	하이투자증권	1786	3546	2953	NNP	*	T	하이투자증권	*	*	*	*	*
...	...	...	...	...	...	...	...	...	...	...	...	...	...
12403	大공개	1786	3546	2953	NNP	*	T	大공개	*	*	*	*	*
12404	테마분석	1786	3546	2953	NNP	*	T	테마분석	*	*	*	*	*
12405	찾은	1786	3546	2953	NNP	*	T	찾은	*	*	*	*	*
12406	TOP	1786	3546	2953	NNP	*	T	TOP	*	*	*	*	*
12407	하기	1786	3545	3780	NNP	*	F	하기	*	*	*	*	*

- 업로드 되는 과정에서 우선순위가 자동으로 생성
- 한자리 수(0~2)로 표현되어야 하는 우선순위가 네자리 수로 표현되어 수정 필요

4. Mecab 사용자 정의사전 구축 프로세스

f) 우선순위 표현 수정

```
for i in user_nnp.대우:
  if len(i)>=4:
    user_nnp.loc[(user_nnp.대우 == i),'3821'] = 0

  elif len(i) == 3:
    user_nnp.loc[(user_nnp.대우 == i),'3821'] = 1

  elif len(i) == 2:
    user_nnp.loc[(user_nnp.대우 == i),'3821'] = 2
```

g) 사전 컴파일

```
!make clean
!make install
```

h) Mecab 함수 정의

```
mecab=Mecab()
```

완성된 mecab 사용자 정의 사전

대우	1786	3545	3821	NNP	*	F	대우.1	*	*.2	*.3	*.4	*.5
구글	1786	3546	2	NNP	*	T	구글	*	*	*	*	*
에이프로젠	1786	3546	0	NNP	*	T	에이프로젠	*	*	*	*	*
의료현장	1786	3546	0	NNP	*	T	의료현장	*	*	*	*	*
목표주가	1786	3545	0	NNP	*	F	목표주가	*	*	*	*	*
하이투자증	1786	3546	0	NNP	*	T	하이투자증	*	*	*	*	*
물류창고	1786	3545	0	NNP	*	F	물류창고	*	*	*	*	*
불온서적	1786	3546	0	NNP	*	T	불온서적	*	*	*	*	*
심의의결	1786	3546	0	NNP	*	T	심의의결	*	*	*	*	*
미시간주	1786	3545	0	NNP	*	F	미시간주	*	*	*	*	*
산업단지	1786	3545	0	NNP	*	F	산업단지	*	*	*	*	*
조회공시	1786	3545	0	NNP	*	F	조회공시	*	*	*	*	*
미래전략실	1786	3546	0	NNP	*	T	미래전략실	*	*	*	*	*
개인투자자	1786	3545	0	NNP	*	F	개인투자자	*	*	*	*	*
주한미군사	1786	3546	0	NNP	*	T	주한미군사	*	*	*	*	*
가온차트	1786	3545	0	NNP	*	F	가온차트	*	*	*	*	*
공공서비스	1786	3545	0	NNP	*	F	공공서비스	*	*	*	*	*
무지갯빛	1786	3546	0	NNP	*	T	무지갯빛	*	*	*	*	*
우대금리	1786	3545	0	NNP	*	F	우대금리	*	*	*	*	*
특별지위	1786	3545	0	NNP	*	F	특별지위	*	*	*	*	*
행정안전부	1786	3545	0	NNP	*	F	행정안전부	*	*	*	*	*
인천지역	1786	3546	0	NNP	*	T	인천지역	*	*	*	*	*

```
print(mecab.pos('2020년 한국 TV 2대중 1대 인터넷 연결된다'))
```

```
[('2020년', 'NNP'), ('한국', 'NNP'), ('TV', 'NNP'), ('2대중', 'NNP'), ('1대', 'NNP'), ('인터넷', 'NNP'), ('연결', 'NNG'), ('된다', 'XSV+EC')]
```

5. FastText model

5-1) FastText 설명 및 모델선정이유

- Facebook의 AI Research (FAIR) 연구소에서 만든 **단어 임베딩 및 텍스트 분류 학습을 위한 라이브러리**
- 294 개 언어로 **사전 학습 된 모델**을 제공
- 단어 임베딩에 신경망을 사용
- **모델 선정 이유**: 등장빈도 수가 낮고 Out Of Vocabulary인 단어들에 대한 효율적인 학습과 예측 속도의 향상을 위해 해당 모델 사용

```
input             # training file path (required)
lr                # learning rate [0.1]
dim               # size of word vectors [100]
ws                # size of the context window [5]
epoch             # number of epochs [5]
minCount          # minimal number of word occurences [1]
minCountLabel     # minimal number of label occurences [1]
minn              # min length of char ngram [0]
maxn              # max length of char ngram [0]
neg               # number of negatives sampled [5]
wordNgrams        # max length of word ngram [1]
loss              # loss function {ns, hs, softmax, ova} [softmax]
bucket            # number of buckets [2000000]
thread            # number of threads [number of cpus]
lrUpdateRate      # change the rate of updates for the learning rate [100]
t                 # sampling threshold [0.0001]
label             # label prefix ['__label__']
verbose           # verbose [2]
pretrainedVectors # pretrained word vectors (.vec file) for supervised learning []
```

- 지도학습 모델링의 경우 사용되는 fastText 모델의 파라미터

5. FastText model

5-2) 모델 학습 및 예측 프로세스

a)

```
# train csv와 test csv를 행 방향으로 concat

train_test = pd.concat([train,test],axis=0)
train_test.reset_index(drop=True, inplace=True)
```

```
# train_test 데이터프레임 중, content열만 추출

X_data = train_test['content']
```

b)

```
# 토큰화한 content의 불필요한 텍스트를 제거

train_token = list1[:len(train)]

for i in range(len(train_token)):
  train_token[i] = str(train_token[i]).strip('[]')
  train_token[i] = str(train_token[i]).replace(',','')
  train_token[i] = str(train_token[i]).replace("'", "")
```

c)

```
# mecab을 사용하여 content를 토큰화

list1 = []

for text in X_data:
    token = mecab.morphs(text)
    list1.append(token)
```

d)

```
# fastText 모델이 label을 인식할 수 있도록 전처리 작업

for i in range(len(train)):
  train['info'][i] = '__label__'+str(train['info'][i])
```

```
# 토큰화 된 content와 전처리된 label을 합쳐서 최종 훈련데이터 구축

train_final = pd.concat([pd.DataFrame(train_token), train['info']],axis=1)
```

e)

```
# 파라미터 튜닝을 위해 train과 validation으로 split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train_final[0], train_final[1], test_size=0.2, random_state=42)

train_final_t = pd.concat([X_train, y_train],axis=1)
train_final_v = pd.concat([X_test, y_test],axis=1)
```

5. FastText model

5-2) 모델 학습 및 예측 프로세스

a)

```
# fastText 모델링 및 파라미터 자동 최적화

!./fasttext supervised -input ./fasttext_t.txt -output model -autotune-validation ./fasttext_v.txt
```

```
Progress: 100.0% Trials:    9 Best score:  0.988463 ETA:   0h 0m 0s
Training again with best arguments
Read 2M words
Number of words:  40837
Number of labels: 2
Progress: 100.0% words/sec/thread:  134267 lr:  0.000000 avg.loss:  0.006094 ETA:   0h 0m 0s
```

최적 파라미터 튜닝 결과, Best score: 약 0.98

b)

```
# test csv 예측 시작 준비
# 이전에 토큰화 한 것 중(list1)에서 test csv의 해당하는 부분 추출

test_token = list1[len(train):]
```

c)

```
# 모델에 입력하기 전, 불필요한 텍스트 전처리

for i in range(len(test_token)):
  test_token[i] = str(test_token[i]).strip('[]')
  test_token[i] = str(test_token[i]).replace(',','')
  test_token[i] = str(test_token[i]).replace("'", "")
```

d)

```
# 튜닝된 fastText 모델로 test csv 예측

!./fasttext predict model.bin ./fasttext_test.txt > result.txt
```

본 팀의 모델을 통해 예측한 내용을 **Dacon에서 최종 채점한 결과, 정확도: 약 97%**

6. 결론 및 개선점

- **결론**
  - 인터넷 기사의 진위여부를 가릴 수 있는 자연어 처리 모델 개발
  - 파라미터 튜닝을 통해, 정확도 97%를 상회하는 안정적인 성능 확보

- **개선점**
  - 외부인터넷 기사를 추가 확보하여 학습에 활용하면, 모델의 문맥 분석력을 높일 수 있을 것으로 기대
  - GloVe를 활용한 워드 임베딩을 fastText 모델에 적용하면, 모델의 성능을 높일 수 있을 것으로 기대

참고사이트

- **soynlp**

https://github.com/lovit/soynlp

https://www.slideshare.net/kimhyunjoonglovit/pycon2017-koreannlp

- **mecab**

https://github.com/hephaex/mecab-ko

https://groups.google.com/g/eunjeon