# 통계적 머신러닝 (STAT424) 최종 프로젝트 레포트

# [ML 13장 감성분석을 활용한 뉴스 기사 분류기]

**2014150353**

**통계 임형준**

## 1. 서론

'감성분석' 방법론은 미리 labeling(긍정적인 내용은 1 그렇지 않으면 0) 된 text 데이터에 기계 학습을 통해 학습된 모델의 활용하여 labeling 되지 않은 새로운 text 데이터의 label을 예측(predict)하는 지도 학습의 일종이다. 여기서 label은 두 가지 밖에 없기 때문에 예측기는 binary classifier가 된다. 만약 같은 방법론을 차용하여 label 종류의 수를 늘리기만 한다면 multi-class classifier를 만들 수 있다. 이러한 발상에 착안하여 뉴스 기사에 대한 장르(정치, 경제, 스포츠 등)를 예측하는 모델을 만들어 보기로 하였다.

뉴스 기사는 네이버(https://news.naver.com)의 기사를 텍스트 파일 형태로 크롤링(crawling)해 사용하고 예측기를 만드는 방법론으로는 Logistic Regression, K-Nearest-Neighbors, Support Vector Machine, Random Forest 등을 활용한다. Grid-Search-Cross-Validation 방법론을 활용하여 각 모델에서 가장 이상적인 hyper-parameter를 찾는다. 그렇게 만들어진 모델들의 예측 정확도(Prediction Accuracy)를 상호 비교하여 가장 성능이 좋은 classifier를 정하도록 한다.

## 2. 크롤링

```
#naver news crawling
from bs4 import BeautifulSoup
import urllib.request
```

크롤링은 python의 BeautifulSoup library를 활용하여 실행하였다. urllib은 http request를 보내고 response를 받기 위한 도구로서 import하였다.

```
# 크롤링 함수
def get_text(URL):
    source_code_from_URL = urllib.request.urlopen(URL)
    soup = BeautifulSoup(source_code_from_URL, 'lxml', from_encoding='utf-8')
    text = ''
    for item in soup.find_all('div', id='articleBodyContents'):
        text = text + str(item.find_all(text=True))
    return text
```

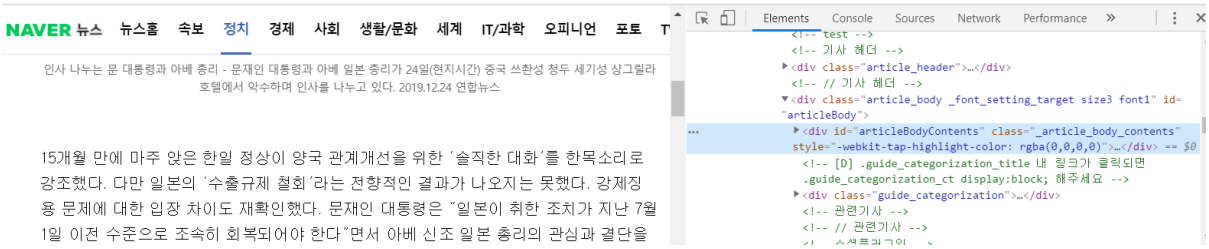
이 사용자 정의 함수는 원하는 URL주소를 입력하였을 경우 해당 주소로 http request를 보내고 response를 받은 후 해당 뉴스 기사에 대한 내용을 BeautifulSoup library를 활용해 파싱하여 text라는 객체에 문자로서 저장하는 역할을 한다. 즉, 네이버 뉴스의 주소를 input으로 입력하면 해당 기사에 대한 text 내용이 output으로 나오게 되는 것이다.[1]

```
for item in soup.find_all('div', id='articleBodyContents'):
    text = text + str(item.find_all(text=True))
```

바로 위의 사용자 정의 함수의 마지막 부분인 soup.find_all()에는 HTTP 요소를 원하는 부분만 가지고 와야 한다. 해당 HTTP 요소는 다음과 같은 방법으로 얻는다.

'https://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=100&oid=081&aid=0003053599'

위의 URL 주소에 해당하는 네이버 뉴스 기사에 들어간 후 구글 크롬의 개발자 도구를 사용하여 기사의 본문에 해당되는 HTTP 태그를 찾아내었다.

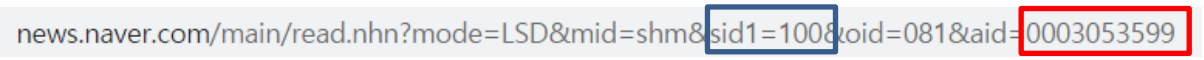



본문에 해당하는 내용은 'div' class 하에 속하는 id='articleBodyContents'에 담겨있음을 알 수 있다. 해당 class의 id를 가져오면 URL 페이지의 많은 내용 중 본문에 해당하는 내용만 정확히 가져올 수 있다.

다음으로 원하는 장르별 기사의 URL 주소를 가져오기 위해 네이버 뉴스의 URL 주소가 구성 되는 구조와 원리를 살펴보자. 아래와 같은 URL 주소의 예시를 보면 주소 구성 원리에 대한 몇 가지 특성을 알아낼 수 있다.

---

[1] (https://yoonpunk.tistory.com/4, [네이버 뉴스 크롤링하기] 참조)




먼저 파란색 상자에 해당하는 부분은 기사의 장르를 특정하는 부분이다. 100부터 103까지는 차례대로 각각 [정치, 경제, 사회, 문화]에 해당하는 기사를 담고 있다. 그리고 붉은 상자에 해당하는 부분은 앞에서 특정된 기사의 장르에서 해당 기사의 누적 기사번호로 추정된다. 이는 붉은 상자에 해당하는 번호에서 100을 뺀 숫자를 넣어도 같은 장르의 다른 과거 기사가 나오는 것에서 알 수 있었다. 이러한 구조에 착안하여 다음과 같이 네 개의 뉴스 기사 장르 별로 각각 100개의 기사를 크롤링할 수 있도록 400개의 url 주소를 저장한 'URL' numpy array를 생성하였다.

```
# URL 주소 [정치]
URL1 = np.array([])
for i in np.arange(100):
  num = 3053599 - i
  new_URL  = np.array(['https://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=100&oid=081&aid=000{}'.format(num)])
  URL1 = np.concatenate((URL1,new_URL))

# URL 주소 [경제]
URL2 = np.array([])
for i in np.arange(100):
  num = 2962982 - i
  new_URL  = np.array(['https://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=101&oid=025&aid=000{}'.format(num)])
  URL2 = np.concatenate((URL2,new_URL))

# URL 주소 [사회]
URL3 = np.array([])
for i in np.arange(100):
  num = 7876 - i
  new_URL  = np.array(['https://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=102&oid=629&aid=000000{}'.format(num)])
  URL3 = np.concatenate((URL3,new_URL))

# URL 주소 [문화]
URL4 = np.array([])
for i in np.arange(100):
  num = 4542990 - i
  new_URL  = np.array(['https://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=103&oid=018&aid=000{}'.format(num)])
  URL4 = np.concatenate((URL4,new_URL))

URL = np.array([URL1, URL2, URL3, URL4])
# 12/25 01:30 AM 기준
```

각 장르별 뉴스 기사의 URL 주소를 식별 번호만 최신 기사(2019/12/25 AM 01:30 기준)에서 최신 기사로부터 100번째 전 기사까지 바꿔가며 각 네 개의 numpy array(100개의 주소 저장)에 저장한 후 하나의 numpy array(총 400개의 URL 주소 저장)에 일괄적으로 저장하였다. 이런 과정을 거쳐 'URL'로 객체화 된 numpy array에는 [정치, 경제, 사회, 문화]별 뉴스 기사의 URL 주소가 각각 100개씩 저장되었다.

앞서 정의한 get_text 크롤링 함수와 url 주소를 담고 있는 URL numpy array를 활용하여 text 데이터를 불러오고 해당 기사에 대응되는 장르의 범주[정치, 경제, 사회, 문화]

또한 지정하여 pandas DataFrame 형태로 객체화 하여 저장한다.

```
#데이터 프레임으로 만들기
import pandas as pd

genre = ['정치','경제','사회','문화']
text = np.array([])
category = np.array([])
for i in np.arange(4):
  for j in np.arange(100):
    cat = np.array([genre[i]])
    category = np.concatenate((category, cat))
    news = np.array([clean(get_text(URL[i][j]))])
    text = np.concatenate((text, news))

data0 = pd.DataFrame(np.array([text, category]), index=['text','category'])
data0 = data0.T
```

붉은 색으로 표시한 부분(clean)은 추후 설명할 데이터 정제(불용어 처리) 사용자 정의 함수이다. 이렇게 data0란 이름으로 객체화 된 pandas DataFrame은 다음과 같은 형태를 띄고 있다.

| | text | category |
|---|---|---|
| 0 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 정치 |
| 1 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 정치 |
| 2 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 정치 |
| 3 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 정치 |
| 4 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 정치 |
| ... | ... | ... |
| 395 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 문화 |
| 396 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 문화 |
| 397 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 문화 |
| 398 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 문화 |
| 399 | 본문 내용 플레이어 플레이어 오류를 우회하기 위한 함수 추가... | 문화 |

400 rows × 2 columns

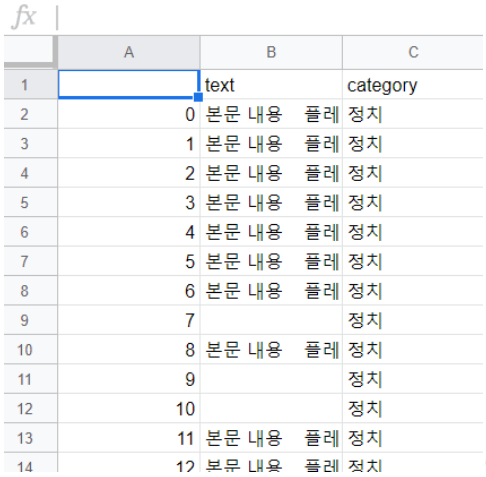
## 2. 결측치 처리

data0로 객체화한 기사를 google drive에 csv화 하여 저장한 후 상태를 살펴보았다.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# csv 파일로 저장
data0.to_csv("data0.csv")
```

[구글 드라이브 마운트 후 csv 파일로 저장]



| | A | B | C |
|---|---|---|---|
| 1 | | text | category |
| 2 | 0 | 본문 내용 플레 | 정치 |
| 3 | 1 | 본문 내용 플레 | 정치 |
| 4 | 2 | 본문 내용 플레 | 정치 |
| 5 | 3 | 본문 내용 플레 | 정치 |
| 6 | 4 | 본문 내용 플레 | 정치 |
| 7 | 5 | 본문 내용 플레 | 정치 |
| 8 | 6 | 본문 내용 플레 | 정치 |
| 9 | 7 | | 정치 |
| 10 | 8 | 본문 내용 플레 | 정치 |
| 11 | 9 | | 정치 |
| 12 | 10 | | 정치 |
| 13 | 11 | 본문 내용 플레 | 정치 |
| 14 | 12 | 본문 내용 플레 | 정치 |

(...후략)

부분적으로 text열에서 결측치가 발견되었다. 따라서 결측치 처리를 진행하였다. 목표가 categorizing인 만큼 근접 item 중 같은 장르의 기사를 포함한 관측치의 text를 그대로 가지고 와 결측치 대체를 시행하였다. (바로 전 기사의 text파일로 대체)

```
# 결측값 대체
# text가 결측일 경우 전 기사의 text 가져오기

for i in np.arange(400):
  if str(data0['text'][i]) == 'nan':
    data0['text'][i] = data0['text'][i-1]
```

[결측치 보완이 끝난 data0 생성 완료/다른 장르의 기사로 대체되는 경우는 없음]

이렇게 성공적으로 결측치 없는 뉴스기사 400개를 pandas DataFrame 형태로 저장하는 것에 성공하였다. 하지만 처음 get_text 함수로 가져온 text는 의미를 담고 있지 않은 불용어를 비롯하여 위의 data0의 결과에서 보이는 바와 같이 공통적으로 모든 기사에 포함된 필요 없는 text(본문 내용~)를 포함하고 있기 때문에 데이터 정제 과정을 진행하였다.

### 3. 데이터 클리닝

```
# 데이터 클리닝 함수
import re

#클린 함수 (영어 및 특수 기호 제거)
def clean(before):
  after = re.sub('[a-zA-Z]', '', before)
  after = re.sub('[₩{₩}₩[₩]₩/?.,;:|₩)*~`!^₩-_+<>@₩#$%&₩₩₩=₩(₩'₩"]','', after)
  return after
```

clean 함수는 text를 before 상태(불용어 포함)에서 after 상태(불용어 제거)로 바꾸는 역할을 한다. 이 함수는 'URL' numpy array와 get_text 함수를 활용하여 'data0'

DataFrame을 만들 당시 과정에서 이미 사용되었다.

```
# 추가 데이터 정제

def addclean(text):
  text = re.sub('[0-9]','',text)
  text = re.sub('▶','',text)
  text = re.sub('■','',text)
  text = re.sub('△','',text)
  text = re.sub('◇','',text)
  text = re.sub('ⓒ', '', text)
  text = re.sub('₩s본문','', text)
  text = re.sub('₩s내용','', text)
  text = re.sub('₩s오류를','', text)
  text = re.sub('₩s플레이어','', text)
  text = re.sub('₩s함수','', text)
  text = re.sub('₩s우회하기','', text)
  text = re.sub('₩s함수','',text)
  text = re.sub('₩s위한','', text)
  text = re.sub('₩s추가','', text)
  return text

for i in np.arange(400):
  data0['text'][i] = addclean(data0['text'][i])
```

추가적인 몇 개의 특수기호와 모든 기사의 첫 부분에 공통적으로 나오는 '본문 내용~' 부분을 없애기 위한 addclean 사용자 정의 함수를 만들어 'data0'의 text 열에 적용하였다. 그 결과는 다음과 같다.

```
data0
```

| | text | category |
|---|---|---|
| 0 | 중국 청두서 文아베 한일 정상회담 서울신문 개월 만에 ... | 정치 |
| 1 | 시니어 헬스케어 로봇과 챗봇 공동연구 서울신문 가천대와... | 정치 |
| 2 | 서울신문 일 경기 용인시 기흥구의 한 단독주택에서 불이나... | 정치 |
| 3 | 서울신문 성탄절 맞아 어린이 환자들에게 선물하는 김정숙 ... | 정치 |
| 4 | 서울신문 손석희 대표이사 사장뉴스 손석희 대표이사 사... | 정치 |

(...후략)

[특수 기호와 '본문 내용~' 공통 부분이 사라진 text열]

**4. Train set & Test set 나누기**

전처리가 끝난 데이터('data0' pandas DataFrame)를 분석에 활용하기 위해 target 변수(y)와 feature 변수(X)로 나누는 작업을 먼저 진행하였다.

먼저 scikit-learn의 전처리 도구 중 하나인 LabelEncoder를 활용하여 data0의 category 부분의 내용을 factor 변수로 변환하여 target 변수 y에 객체화 하여 저장했다.

```
# target에 숫자로 label 부여

from sklearn.preprocessing import LabelEncoder
labeling = LabelEncoder()
y = labeling.fit_transform(data0['category'].values)
y
```

```
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. (...후략)
```

'data0'의 text의 경우 이미 전처리가 완료된 상황이기 때문에 바로 feature변수 X에 객체화하여 저장하였다.

```
# feature variable X 생성
X = data0['text'].values
```

이제 scikit-learn의 train_test_split을 import하여 training set과 test set으로 나누는 작업을 실시한다. 이 때, stratify = y 옵션을 사용하여 기사 장르별 상대적 비중이 나눠진 set에도 동일하도록 설정하였다. test_size=0.3을 통해 test set이 전체 데이터의 30%를 가져가도록 설정하였다.

```
# train, test set 나누기

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)
```

**5. LogisticRegression, KNeighborsClassifier, SVC, RandomForestClassifier를 통한 예측**

준비된 training set과 test set에 서론에 명시한 네 가지 방법을 사용하여 모델을 학습시킨 뒤 test_set에 대한 category prediction과 실제 category의 비교를 통해 prediction accuracy를 구해보고 분류기로서 가장 적절한 모델을 선정해본다.

또한, 같은 방법론 안에서도 GridSearchCV를 활용하여 가장 성능이 좋은 모델을 만드는 초모수(hype-parameter)의 조합을 알아보도록 하였다. 다음은 각 방법론을 활용하여 구현한 모델의 코드, 가장 성능 좋은 초모수 조합, training set과 test set에 대한 prediction accuracy 그리고 confusion matrix이다.

## 5-1 Logistic Regression

### code)

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(strip_accents=None,lowercase=False,preprocessor=None)
param_grid = [{'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer,None],'clf__penalty':['l1', 'l2'],'clf__C': [1.0, 10.0, 100.0]},
              {'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer, None],'vect__use_idf':[False],
               'vect__norm':[None], 'clf__penalty': ['l1', 'l2'],'clf__C': [1.0, 10.0, 100.0]}]
lr_tfidf = Pipeline([('vect', tfidf),('clf', LogisticRegression())])
gs_lr_tfidf = GridSearchCV(lr_tfidf, param_grid,scoring='accuracy',cv=5, verbose=1,n_jobs=1)
gs_lr_tfidf.fit(X_train, y_train)
```

### best hyper-parameters )

```
print(gs_lr_tfidf.best_params_)
```

```
{'clf__C': 1.0, 'clf__penalty': 'l1', 'vect__ngram_range': (1, 1), 'vect__tokenizer':
```

### prediction accuracy[train, test 순서] )

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_train, y_train_pred1)) # train data에 대한 accuracy
print(accuracy_score(y_test, y_test_pred1)) # test data에 대한 accuracy
```

```
1.0
1.0
```

### confusion matrix )

```
# 분류 결과 logistic
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_test_pred1))
```

```
[[30  0  0  0]
 [ 0 30  0  0]
 [ 0  0 30  0]
 [ 0  0  0 30]]
```

---

## 5-2 K-Nearest-Neighbors

### code)

```
from sklearn.neighbors import KNeighborsClassifier
tfidf = TfidfVectorizer(strip_accents=None,lowercase=False,preprocessor=None)
param_grid = [{'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer,None],'knn__n_neighbors':[3,5,10],'knn__p': [1,2]},
              {'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer, None],'vect__use_idf':[False],
               'vect__norm':[None], 'knn__n_neighbors':[3,5,10],'knn__p': [1,2]}]
knn_tfidf = Pipeline([('vect', tfidf),('knn', KNeighborsClassifier())])
gs_knn_tfidf = GridSearchCV(knn_tfidf, param_grid,scoring='accuracy',cv=5, verbose=1,n_jobs=1)
gs_knn_tfidf.fit(X_train, y_train)
```

### best hyper-parameters )

```
print(gs_knn_tfidf.best_params_)
```

```
{'knn__n_neighbors': 10, 'knn__p': 2, 'vect__ngram_range': (1, 1), 'vect__tokenizer': None}
```

### prediction accuracy[train, test 순서] )

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_train, y_train_pred2)) # train data에 대한 accuracy
print(accuracy_score(y_test, y_test_pred2)) # test data에 대한 accuracy
```

```
0.9
0.8
```

### confusion matrix )

```
# 분류 결과 KNN
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_test_pred2))
```

```
[[17  1  6  6]
 [ 0 27  0  3]
 [ 1  1 26  2]
 [ 0  2  2 26]]
```

## 5-3 Support-Vector-Classifier

### code)

```
from sklearn.svm import SVC
tfidf = TfidfVectorizer(strip_accents=None,lowercase=False,preprocessor=None)
param_grid = [{'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer,None],'svc__C':[1e3,5e3,1e4,5e4,1e5],
               'svc__gamma': [0.005,0.01,0.1], 'svc__kernel':['poly','rbf','sigmoid']},
              {'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer, None],'vect__use_idf':[False],
               'vect__norm':[None],'svc__C':[1e3,5e3,1e4,5e4,1e5],
               'svc__gamma': [0.005,0.01,0.1], 'svc__kernel':['poly','rbf','sigmoid']}]
svc_tfidf = Pipeline([('vect', tfidf),('svc', SVC())])
gs_svc_tfidf = GridSearchCV(svc_tfidf, param_grid,scoring='accuracy',cv=5, verbose=1,n_jobs=1)
gs_svc_tfidf.fit(X_train, y_train)
```

### best hyper-parameters )

```
print(gs_svc_tfidf.best_params_)
```

```
{'svc__C': 1000.0, 'svc__gamma': 0.005, 'svc__kernel': 'rbf', 'vect__ngram_range': (1, 1), 'vect__tokenizer': None}
```

### prediction accuracy[train, test 순서] )

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_train, y_train_pred3)) # train data에 대한 accuracy
print(accuracy_score(y_test, y_test_pred3)) # test data에 대한 accuracy
```

```
1.0
0.95
```

### confusion matrix )

```
# 분류 결과 SVC
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_test_pred3))
```

```
[[28  0  1  1]
 [ 0 30  0  0]
 [ 1  0 28  1]
 [ 0  2  0 28]]
```

## 5-4 Random Forest Classifier

### code)

```
from sklearn.ensemble import RandomForestClassifier
tfidf = TfidfVectorizer(strip_accents=None,lowercase=False,preprocessor=None)
param_grid = [{'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer,None],
               'rf__n_estimators':[50,100,200,500],'rf__max_depth': [3,5,10]},
              {'vect__ngram_range': [(1,1)],'vect__tokenizer': [tokenizer, None],'vect__use_idf':[False],
               'vect__norm':[None],'rf__n_estimators':[50,100,200,500],'rf__max_depth': [3,5,10]}]
rf_tfidf = Pipeline([('vect', tfidf),('rf', RandomForestClassifier())])
gs_rf_tfidf = GridSearchCV(rf_tfidf, param_grid,scoring='accuracy',cv=5, verbose=1,n_jobs=1)
gs_rf_tfidf.fit(X_train, y_train)
```

### best hyper-parameters )

```
{'rf__max_depth': 3, 'rf__n_estimators': 50,

 'vect__ngram_range': (1, 1), 'vect__tokenizer': <function tokenizer at 0x7faa73d44620>}
```

**prediction accuracy[train, test 순서] )**

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_train, y_train_pred4)) # train data에 대한 accuracy
print(accuracy_score(y_test, y_test_pred4)) # test data에 대한 accuracy
```

```
1.0
1.0
```

**confusion matrix )**

```
# 분류 결과 RandomForest
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_test_pred4))
```

```
[[30  0  0  0]
 [ 0 30  0  0]
 [ 0  0 30  0]
 [ 0  0  0 30]]
```

---

결과를 정리하자면 test set에 대한 prediction accuracy 기준으로 볼 때, Logistic Regression과 Random Forest의 결과가 가장 좋았고 그 다음으로 좋은 성능을 보인 것은 Support-Vector-Classifier이다. K-Nearest-Classifier는 다른 방법론을 사용한 모델에 비해 성능이 다소 부진한 모습을 보였다.

## 6. 한계점 및 의의

비록 결과가 나오고 성능이 좋은 분류기까지 선별하였지만 아쉬운 점은 여전히 남는다. 이번 프로젝트의 한계점에 대해 알아보았다.

1. 100개의 서로 다른 기사를 활용하지 못하였다.
    - HTML 요소에 대한 이해나 URL 주소의 생성 원리에 대한 이해가 부족해 크롤링 과정에서 결측이 발생하였다. 비록 같은 범주의 다른 text로 대체하였지만 표본의 정보는 결측 상태 그대로이므로 결측값 대체로서의 의미가 퇴색된다.
2. 표본이 적어 Logistic Regression 분류기와 Random Forest 분류기의 우열을 가리지 못하였다.
    - 비록 기사를 장르별로 100개씩 활용하였지만 실제로는 하루에 쏟아져 나오는 기사의 양만 고려해봐도 모든 새로운 기사에 대한 분류기로 활용되기에는 턱없이 부족한 표본 숫자이다.
3. 좀 더 심화된 방법론이나 앙상블 학습을 활용하지 않았다.
    - 머신러닝에 대한 좀 더 깊은 이해와 높은 수준의 코딩 능력이라면 더 큰 표본을 활용한 다양한 앙상블학습을 시도해봤어야 한다.

그럼에도 불구하고 데이터 수집부터 전처리와 모델링 및 구현까지 데이터 분석 과정을 처음부터 끝까지 해본 것에 의미가 있었던 프로젝트였다.