In [ ]:

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

In [ ]:

```
import numpy as np
import pandas as pd
```

In [ ]:

```
# 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
```

In [ ]:

```
URL1[0]
```

Out[ ]:

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

In [ ]:

```
# 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
```

In [ ]:

```
# 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
```

In [ ]:

```
#데이터 프래임으로 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
```

In [ ]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [ ]:

```
%pwd
```

Out[ ]:

```
'/content/drive/My Drive'
```

In [ ]:

```
%cd /content/drive/My Drive
```

```
/content/drive/My Drive
```

In [ ]:

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

In [ ]:

```
 #csv 파일 불러오기
 data0 = pd.read_csv('data0.csv')
```

In [ ]:

```
data0 = data0.loc[:,['text','category']]
data0
```

Out[ ]:

	text	category

	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
...	...	...
395	국민은행 노조위원장 시절 파업 주도 임명강행시 여당과 ...	문화
396	이데일리 김대웅 기자 넥스트사이언스는 자회사 엘리샤코이가...	문화
397	이데일리 현재 코스닥 외국인 억 순매도 마켓포인트 ...	문화
398	이데일리 현재 코스닥 개인 억 순매도 마켓포인트 네...	문화
399	이데일리 현재 코스닥 기관 억 순매수 마켓포인트 네...	문화

400 rows × 2 columns

In [ ]:

```
data0
```

Out[ ]:

	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
...	...	...
395	국민은행 노조위원장 시절 파업 주도 임명강행시 여당과 ...	문화
396	이데일리 김대웅 기자 넥스트사이언스는 자회사 엘리샤코이가...	문화
397	이데일리 현재 코스닥 외국인 억 순매도 마켓포인트 ...	문화
398	이데일리 현재 코스닥 개인 억 순매도 마켓포인트 네...	문화
399	이데일리 현재 코스닥 기관 억 순매수 마켓포인트 네...	문화

400 rows × 2 columns

In [ ]:

```
# target에 숫자로 label 부여

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

Out[ ]:

```
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, 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,
       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, 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, 0, 0,
       0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1])
```

In [ ]:

```
# 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]

data0['text'].head(10)
```

Out[ ]:

```
0                    중국 청두서 文아베 한일 정상회담 서울신문 개월 만에 ...
1                    시니어 헬스케어 로봇과 챗봇 공동연구 서울신문 가천대와...
2                   서울신문 일 경기 용인시 기흥구의 한 단독주택에서 불이나...
3                   서울신문 성탄절 맞아 어린이 환자들에게 선물하는 김정숙 ...
4                   서울신문 손석희  대표이사 사장뉴스 손석희  대표이사 사...
5                    '여성 호 치안정감' 이은정 경찰대학장 서울신문 남성영...
6                   서울신문 문재인 대통령과 아베 일본 총리가 일현지시간 중...
7                   서울신문 문재인 대통령과 아베 일본 총리가 일현지시간 중...
8       본문 내용    플레이어     플레이어     오류를 우회하기 위한 함수 추가...
9       본문 내용    플레이어     플레이어     오류를 우회하기 위한 함수 추가...
Name: text, dtype: object
```

In [ ]:

```
# 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])

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

In [ ]:

```
# 문장 vector화 하는 vectorizer

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
text = [data0['text'][7]]
voc = vectorizer.fit_transform(text)
print(vectorizer.get_feature_names())
print(voc.shape)
```

```
['가까운', '가장', '갈등', '강조했다', '같이', '걸어가야', '것이', '계속', '계정에', '공유하는', '과거의', '
구독하기', '국가들과', '국은', '귀국길에', '그러나', '글을', '기자', '긴밀히', '나가야', '나라든', '네이버에
서', '대통령과', '대통령은', '동안의', '때때로', '떠나며', '라는', '마치고', '모두', '무단전재', '무료만화',
'문재인', '문화를', '미래지향적인', '발전시켜', '발전해', '밝혔다', '방중', '보기', '분명히', '불거지는', '불
행한', '사는', '샹그릴라호텔에서', '서울신문', '세기성', '세상에', '소셜네트워크서비스', '속에서', '수천', '쓰
```

찬성', '아베', '악수하고', '앞서', '어느', '어울려', '언급했다', '없다', '역사로', '역사를', '역사와', '연합
뉴스', '오랜', '오른', '올려', '요소가', '우리가', '우리는', '이날', '이라고', '이라며', '이라면서', '이런',
'이렇게', '이웃', '인해', '일본', '일이', '일정을', '일현지시간', '있다', '자신의', '재배포금지', '정상회담에
', '제목의', '중국', '직시하면서도', '청두', '청두를', '총리가', '최선을', '클릭', '하고', '한다', '한중일',
'함께', '협력', '협력을', '협력해야', '홀로']
(1, 101)

In [ ]:

```
# 장르 별 최빈단어 10개씩 찾기

vocab1 = np.array([])
for i in np.arange(100):
  text = np.array([X[i]])
  new_vocab = vectorizer.fit_transform(text)
  vocab1 = np.concatenate((vocab1,vectorizer.get_feature_names()))
```

Out[ ]:

18176

In [ ]:

```
# 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)
```

In [ ]:

```
# 단어를 분할하는 tokenizer  - 안써도 될듯
def tokenizer(text):
    return text.split()
```

In [ ]:

```
#LogisticRegression

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)
```

In [ ]:

```
print(gs_lr_tfidf.best_params_)

y_train_pred1 = gs_lr_tfidf.predict(X_train)
y_test_pred1 = gs_lr_tfidf.predict(X_test)

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
```

{'clf__C': 1.0, 'clf__penalty': 'l1', 'vect__ngram_range': (1, 1), 'vect__tokenizer': <function to
kenizer at 0x7faa73d44620>}
1.0
1.0

In [ ]:

```
# 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]]
```

In [ ]:

```
#KNN

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

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)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 120 out of 120 | elapsed:   14.4s finished
```

Out[ ]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('vect',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=False,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
                                                        norm='l2',
                                                        preprocessor=None,
                                                        smooth_idf=True,
                                                        stop_word...
                                 'vect__ngram_range': [(1, 1)],
                                 'vect__tokenizer': [<function tokenizer at 0x7faa73d44620>,
                                                     None]},
                                {'knn__n_neighbors': [3, 5, 10], 'knn__p': [1, 2],
                                 'vect__ngram_range': [(1, 1)], 'vect__norm': [None],
                                 'vect__tokenizer': [<function tokenizer at 0x7faa73d44620>,
                                                     None],
                                 'vect__use_idf': [False]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

In [ ]:

```
print(gs_knn_tfidf.best_params_)

y_train_pred2 = gs_knn_tfidf.predict(X_train)
```

```
y_test_pred2 = gs_knn_tfidf.predict(X_test)

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
```

```
{'rf__max_depth': 3, 'rf__n_estimators': 50, 'vect__ngram_range': (1, 1), 'vect__tokenizer': <func
tion tokenizer at 0x7faa73d44620>}
0.9
0.8
```

In [ ]:

```
# 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]]
```

In [ ]:

```
#SVM

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

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':[1e3,5e3,1
e4,5e4,1e5],
               '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)
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 900 out of 900 | elapsed:  2.8min finished
```

Out[ ]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('vect',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
                                                        lowercase=False,
                                                        max_df=1.0,
                                                        max_features=None,
                                                        min_df=1,
                                                        ngram_range=(1, 1),
                                                        norm='l2',
                                                        preprocessor=None,
                                                        smooth_idf=True,
                                                        stop_word...
                                                    None]},
                                {'svc__C': [1000.0, 5000.0, 10000.0, 50000.0,
                                            100000.0],
                                 'svc__gamma': [0.005, 0.01, 0.1],
```

```
                        'svc__kernel': ['poly', 'rbf', 'sigmoid'],
                        'vect__ngram_range': [(1, 1)], 'vect__norm': [None],
                        'vect__tokenizer': [<function tokenizer at 0x7faa73d44620>,
                                            None],
                        'vect__use_idf': [False]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

In [ ]:

```
print(gs_svc_tfidf.best_params_)

y_train_pred3 = gs_svc_tfidf.predict(X_train)
y_test_pred3 = gs_svc_tfidf.predict(X_test)

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
```

```
{'svc__C': 1000.0, 'svc__gamma': 0.005, 'svc__kernel': 'rbf', 'vect__ngram_range': (1, 1),
'vect__tokenizer': None}
1.0
0.95
```

In [ ]:

```
# 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]]
```

In [ ]:

```
#Random Forest

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

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)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 240 out of 240 | elapsed:  1.5min finished
```

Out[ ]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('vect',
                                        TfidfVectorizer(analyzer='word',
                                                        binary=False,
                                                        decode_error='strict',
                                                        dtype=<class 'numpy.float64'>,
                                                        encoding='utf-8',
                                                        input='content',
```

```
                                                lowercase=False,
                                                max_df=1.0,
                                                max_features=None,
                                                min_df=1,
                                                ngram_range=(1, 1),
                                                norm='l2',
                                                preprocessor=None,
                                                smooth_idf=True,
                                                stop_word...
                         'vect__ngram_range': [(1, 1)],
                         'vect__tokenizer': [<function tokenizer at 0x7faa73d44620>,
                                             None]},
                        {'rf__max_depth': [3, 5, 10],
                         'rf__n_estimators': [50, 100, 200, 500],
                         'vect__ngram_range': [(1, 1)], 'vect__norm': [None],
                         'vect__tokenizer': [<function tokenizer at 0x7faa73d44620>,
                                             None],
                         'vect__use_idf': [False]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

In [ ]:

```
print(gs_rf_tfidf.best_params_)

y_train_pred4 = gs_rf_tfidf.predict(X_train)
y_test_pred4 = gs_rf_tfidf.predict(X_test)

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
```

```
{'rf__max_depth': 3, 'rf__n_estimators': 50, 'vect__ngram_range': (1, 1), 'vect__tokenizer': <func
tion tokenizer at 0x7faa73d44620>}
1.0
1.0
```

In [ ]:

```
# 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]]
```