Naver News Sentiment Analysis

1조 김연상
진형민 강창민

목 차

프로젝트 목표
핵심 개념

프로젝트 목표 및 핵심 개념

▪프로젝트 목표

▪머신 러닝 훈련 데이터로 7개의 모델을 훈련 시켜 기준이 되는 GridSearchCV best_score_를 바탕으로 상위 3개 모델 추출한 뒤 사용자가 입력한 키워드와 연관된 뉴스 데이터를 크롤링 하여 상위 3개 모델로 뉴스 데이터의 감정을 예측한다.
▪이후 3개 모델의 예측 값이 전부 일치하는지(모두 긍정 or 부정)정확도를 출력한다.

▪핵심 개념

▪1) 감성 분석

▪감성 분석(sentiment analysis)은 텍스트 데이터에서 언어 사용자의 주관적인 의견, 태도, 감정 등을 파악하는 자연어 처리 기술 중 하나로 컴퓨터가 텍스트를 분석하여 긍정, 부정적인 감정을 판단할 수 있음.

▪감성 분석은 주로 머신 러닝, 딥러닝 등의 인공지능 기술을 이용하여 구현되며, 자연어 처리 분야에서 주로 사용됨.

프로젝트 목표 및 핵심 개념

▪핵심 개념

▪2) 특성 벡터화 및 특성 추출

▪머신 러닝 알고리즘으로 텍스트를 분석하기 위해서는 텍스트를 구성하는 단어 기반의 특성 추출을 하고 이를 숫자형 값인 벡터 값으로 표현해야 한다. 자연어 처리에 일반적으로 사용되는 방식으로 BoW(Bag of World)가 있으며 문서에 등장하는 단어의 빈도를 기반으로 문서를 벡터화 하는 방식.

▪각 단어가 문서에서 얼마나 자주 등장하는가에 따라 각 단어의 벡터가 구성되며, 단어의 순서나 문맥 등을 고려하지 않아 문장의 의미나 표현 방식을 완벽하게 반영하지는 못한다는 단점이 있어 확실한 전처리가 필요함.

▪대표적으로 카운트 기반 벡터화와 TF-IDF 기반 벡터화 방식이 있으며, 이번 프로젝트에선 TF-IDF 방식이 사용됨.

프로젝트 목표 및 핵심 개념

▪핵심 개념

▪2-1) 카운트 기반 벡터화

▪사이킷런의 CountVectorizer 모듈에서 제공되며 각 문서에서 단어가 등장하는 빈도를 해당 단어 피처에 숫자값으로 할당됨.

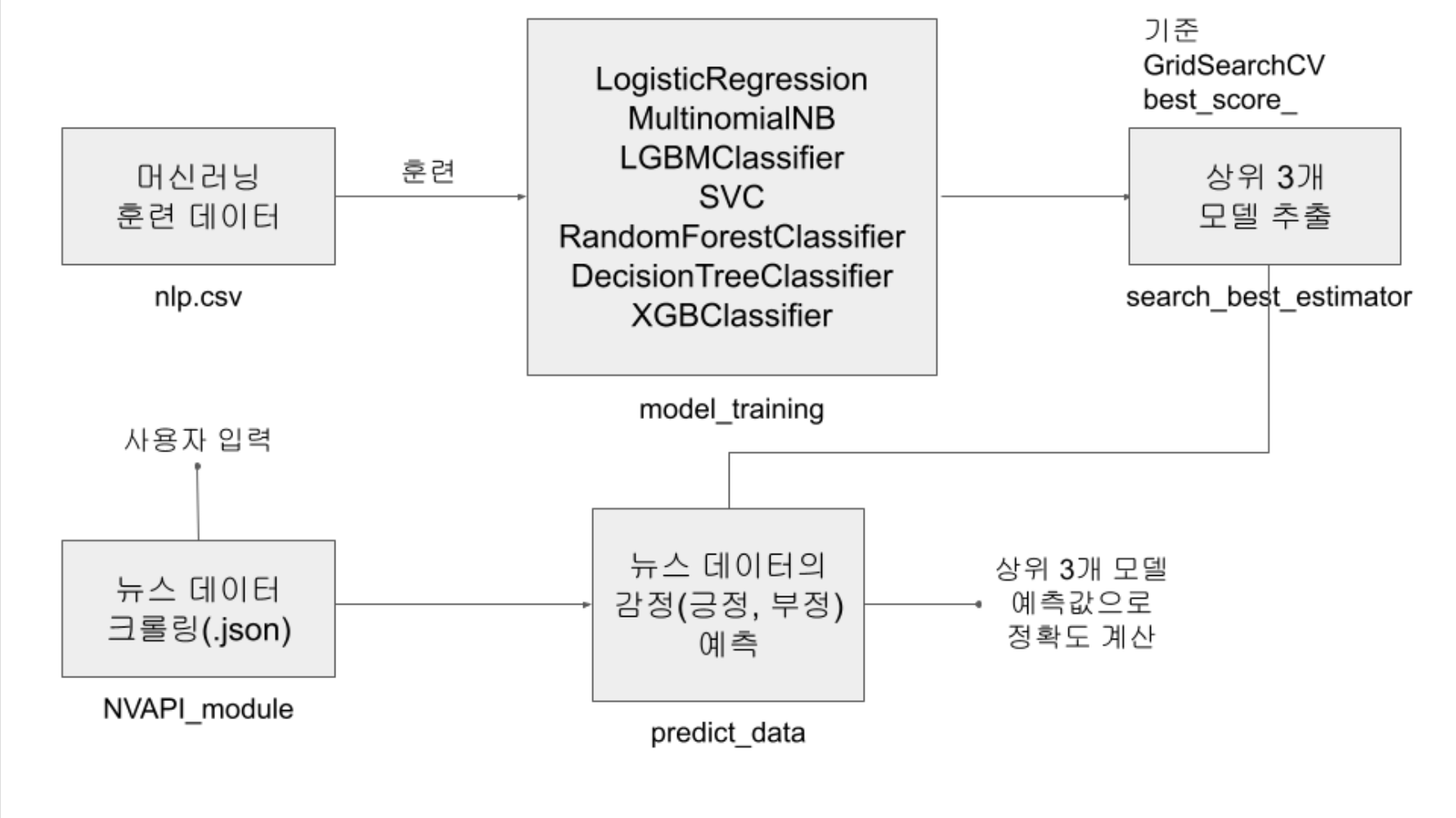
▪각 문서별 단어의 빈도를 정리하여 문서 단어 행렬(DTM)을 구성하며, 단어 출현 빈도가 높을수록 중요도가 높게 다루어지며 문서 d에 등장한 단어 t의 횟수는 tf(t,d)로 표현됨.

▪2-2) TF-IDF 기반 벡터화

▪단어의 빈도 뿐만 아니라 문서 집합 전체의 특이성을 반영하여 문서를 벡터화하는 기법

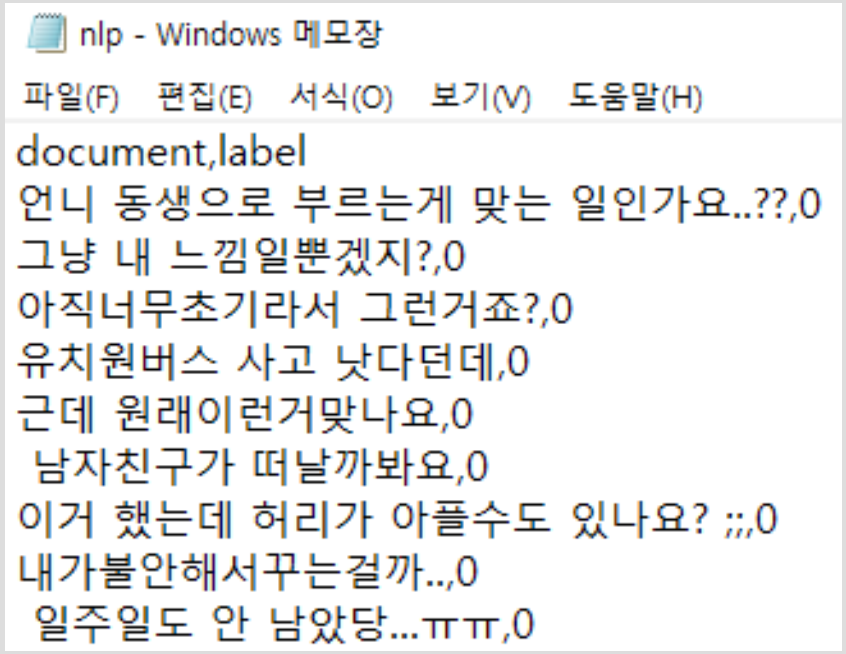
▪단어 빈도(TF, Term Frequency)와 역문서 빈도(IDF, Inverse Document Frequency) 두가지의 지표를 이용하여 벡터화 하는 기법으로, 모든 문서에 자주 등장하는 단어벡터에 대한 가중치를 줄이고 특정 문서만 등장하는 단어벡터에는 가중치를 높이는 방식

프로젝트 구성도


머신러닝
훈련 데이터
nlp.csv
훈련
LogisticRegression
MultinomialNB
LGBMClassifier
SVC
RandomForestClassifier
DecisionTreeClassifier
XGBClassifier
model_training
기준
GridSearchCV
best_score_
상위 3개
모델 추출
search_best_estimator
사용자 입력
뉴스 데이터
크롤링(.json)
NVAPI_module
뉴스 데이터의
감정(긍정, 부정)
예측
predict_data
상위 3개 모델
예측값으로
정확도 계산

머신러닝 데이터 분석 및 정제

훈련용 데이터 분석 및 탐색




```
In [18]: data['label'].value_counts()
Out[18]:
0    7067
1    7067
Name: label, dtype: int64
```

- 프로젝트에서 사용된 감성 분석 데이터인 nlp.csv의 분석
- nlp.csv는 document 및 label 두가지의 컬럼으로 구성됨
- nlp.csv 파일 내 레이블 별 데이터의 갯수는 0 : 7067개, 1:7067개로 동일
- 특수문자등 한글이 아닌 텍스트가 확인됨

데이터 전처리-1

```python
# 데이터를 정제하는 함수
def data_cleansing(data):
    # null값 제거
    df_nlp = data[data['document'].notnull()]

    # 정규식으로 한글만 남김
    df_nlp['document'] = df_nlp['document'].apply(
        lambda x: re.sub(r'[^ㄱ-ㅣ가-힣]+', ' ', x))
    # 1글자 단어 제거
    df_nlp['document'] = df_nlp['document'].apply(
        lambda x: ' '.join([w for w in x.split() if len(w) > 1]))

    return df_nlp
```

```python
# 데이터를 train과 test set으로 분리하는 함수
def data_tts(data, tfidf):
    df_nlp = data

    x_data = df_nlp['document']
    y_data = df_nlp['label']

    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
                                                        test_size=0.2,
                                                        random_state=0,
                                                        stratify=y_data)
    tfidf.fit(x_train)

    train_nlp_tfidf = tfidf.transform(x_train)
    test_nlp_tfidf = tfidf.transform(x_test)

    return train_nlp_tfidf, test_nlp_tfidf, y_train, y_test
```

- 데이터 전처리로 데이터에 존재할 수 있는 Null값 제거
- 정규식(r'[^ㄱ-ㅣ가-힣]+')을 이용하여 'ㄱ'으로 시작하거나 '가'부터 '힣'까지의 문자를 제외한 나머지는 공백으로 치환
- 한글만 남은 데이터에서 불필요한 1글자의 단어들 제거
- 정제된 데이터를 훈련용 데이터와 테스트용 데이터로 분리해주며, tfidf 객체를 받아 transform 하는 함수 생성
- x_data는 감정분석을 위한 문자열 데이터, y_data는 긍정 부정의 label을 담은 데이터로 구성

데이터 전처리-2

```
# 한글을 형태소 별로 분리하는 함수
def okt_tokenizer(text):
    okt = Okt() #konlpy 클래스의 함수
    tokens = okt.morphs(text) # morphs 함수
    return tokens             # 한글문장을 형태소별로 나누어준다
```

- Tfidf 객체 정의할 때, 매개변수로 사용할 tokenizer 함수 생성

- 위 함수는 konlpy 패키지의 okt.morphs() 함수를 이용하여 문장을 형태소단위로 토큰화 해줌

```
# 데이터 정제
data = data_cleansing(data)

# 자연어 데이터를 숫자 데이터로 바꾸기 위해 TfidfVectorizer 사용
# 한글을 형태소 별로 나누기 위해 Okt 사용
tfidf = TfidfVectorizer(tokenizer=okt_tokenizer,
                        ngram_range=(1, 2),  # 단어크기 1~2개 단어
                        min_df=3,            # 출현빈도 최소 3
                        max_df=0.9)          # 너무 높은 출현빈도 제외
# 데이터를 훈련과 검증용으로 나눔
x_train, x_test, y_train, y_test = data_tts(data, tfidf)
```

- tfidf객체 생성 시 매개변수 설정

tokenizer = okt_tokenizer()

okt.morphs 함수를 이용하여 토큰화 시켜주는 함수

ngram_range = 1~2

토큰의 단어크기 1~2개단어로 지정

min_df = 3

토큰의 출연빈도 최소 3번 이상

max_df = 0.9

너무높은 출현빈도의 토큰 제외(90%이상)

- 위에서 정의된 함수들을 이용하여 데이터 정제 코드를 입력하면 Tf-idf기반으로 벡터화된 데이터를 얻을 수 있음

여러가지 모델학습

여러가지 모델을 이용한 감성 분석 머신러닝

```python
# 모델을 훈련시키는 함수
def model_training(model, params, x_train, x_test, y_train, y_test):
    model_name = str(model).split('(')[0]

    grid_cv = GridSearchCV(model, param_grid=params, cv=3,
                           scoring='accuracy', verbose=1, n_jobs=-1)
    grid_cv.fit(x_train, y_train)

    estimator = grid_cv.best_estimator_
    score = grid_cv.best_score_
    pred = estimator.predict(x_test)
    # grid_cv.best_params_ : 좋은 모델 선정 기준
    print("=" * 7, model_name, "=" * 7, "\n", grid_cv.best_params_, "\n== Best Score ==\n", score)
    print("== Train ==\n", estimator.score(x_train, y_train))
    print("== Test ==\n", estimator.score(x_test, y_test), "\n")

    if model_name == "SVC":
        get_clf_eval(y_test, pred, pred)
    else:
        pred_proba = estimator.predict_proba(x_test)[:, 1]
        get_clf_eval(y_test, pred, pred_proba)

    print(classification_report(y_test, pred, target_names=['negative', 'positive']))

    return estimator, score
```

- 각 모델별로 반복하여 학습을 진행하기 때문에 함수화 진행

- 매개변수로 모델명과 params, 훈련, 테스트 데이터를 받아 params에 정의된 매개변수들을 GridSearchCV 방식으로 최적의 파라미터와 그 점수를 반환

Logistic Regression 모델

```
# 모델 생성 후 훈련 및 검증
# LogisticRegression model
model = LogisticRegression(random_state=0, solver='saga')
params = {'C': [2.5, 3.0, 3.5]}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- Logistic Regression 모델을 이용한 분석

- 하이퍼 매개변수 C의 최적값을 구하기 위하여 여러 C값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

Logistic Regression 모델 평가

```
======= LogisticRegression =======
 {'C': 3.0}
=== Best Score ===
 0.8410719023613691
=== Train ===
 0.9401255859202264
=== Test ===
 0.8457729041386629

오차 행렬
[[1225  189]
 [ 247 1166]]
정확도: 0.8458, 정밀도: 0.8605, 재현율: 0.8252,    F1: 0.8425, AUC:0.9277
              precision    recall  f1-score   support

    negative       0.83      0.87      0.85      1414
    positive       0.86      0.83      0.84      1413

    accuracy                           0.85      2827
   macro avg       0.85      0.85      0.85      2827
weighted avg       0.85      0.85      0.85      2827
```

- 최적의 파라미터 C:3.0에서의 훈련 점수 0.94 / 테스트 점수 0.84 확인됨
- AUC 정확도 92.7% / ROC_AUC 정확도 84.1%

Naive_Bayes 모델

```
# NaiveBayes model
model = MultinomialNB()
params = {'alpha': [1.0, 1.5, 2.0]}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- Multinomia Naïve_Bayes 모델을 이용한 분석

- 하이퍼 매개변수 alpha의 최적값을 구하기 위하여 alpha 값 여러 개를 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

Naive_Bayes 모델

```
# NaiveBayes model
model = MultinomialNB()
params = {'alpha': [1.0, 1.5, 2.0]}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- Multinomia Naïve_Bayes 모델을 이용한 분석
- 하이퍼 매개변수 alpha의 최적값을 구하기 위하여 여러 alpha 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

Naive_Bayes 모델 평가

```
======= MultinomialNB =======
 {'alpha': 1.5}
=== Best Score ===
 0.8460245865393118
=== Train ===
 0.9042186256301407
=== Test ===
 0.8546162009197029

오차 행렬
[[1234  180]
 [ 231 1182]]
정확도: 0.8546, 정밀도: 0.8678, 재현율: 0.8365,    F1: 0.8519, AUC:0.9330
              precision    recall  f1-score   support

    negative       0.84      0.87      0.86      1414
    positive       0.87      0.84      0.85      1413

    accuracy                           0.85      2827
   macro avg       0.86      0.85      0.85      2827
weighted avg       0.86      0.85      0.85      2827
```

- 최적의 파라미터 alpha:1.5에서의 훈련 점수 0.90 / 테스트 점수 0.85 확인됨
- AUC 정확도 93.3% / ROC_AUC 정확도 84.6%

LGBM Classifier 모델

```
# LGB
model = LGBMClassifier(random_state=0)
params = {'n_estimators': [200, 500, 750, 1000],
          'learning_rate': [0.05, 0.1, 0.5],
          'max_depth': [3, 5, 7]
          }
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- LGBM Classifier 모델을 이용한 분석

- 하이퍼 매개변수 n_estimators, learning_rate, max_depth 의 최적값을 구하기 위하여 여러 n_estimators, learning_rate, max_depth 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

LGBM Classifier 모델

```
# LGB
model = LGBMClassifier(random_state=0)
params = {'n_estimators': [200, 500, 750, 1000],
          'learning_rate': [0.05, 0.1, 0.5],
          'max_depth': [3, 5, 7]
          }
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- LGBM Classifier 모델을 이용한 분석

- 하이퍼 매개변수 n_estimators, learning_rate, max_depth 의 최적값을 구하기 위하여 여러 n_estimators, learning_rate, max_depth 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

LGBM Classifier 모델 평가

```
======= LGBMClassifier =======
 {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
=== Best Score ===
 0.7704077120367914
=== Train ===
 0.8251525603608384
=== Test ===
 0.7845772904138663

오차 행렬
[[1254  160]
 [ 449  964]]
정확도: 0.7846, 정밀도: 0.8577, 재현율: 0.6822,    F1: 0.7600, AUC:0.8789
              precision    recall  f1-score   support

    negative       0.74      0.89      0.80      1414
    positive       0.86      0.68      0.76      1413

    accuracy                           0.78      2827
   macro avg       0.80      0.78      0.78      2827
weighted avg       0.80      0.78      0.78      2827
```

- 최적의 파라미터 learning_rate : 0.1, max_depth : 5, n_estimators : 500 에서 훈련 점수 0.82 / 테스트 점수 0.78 확인됨

- AUC 정확도 87.8% / ROC_AUC 정확도 77.0%

SVC 모델

```
# SVC model
model = SVC(random_state=0)
params = {'C': [0.1, 0.5, 1],
          'gamma': ['auto', 0.1, 0.01],
          'kernel': ['linear', 'rbf', 'poly']}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- SVC 모델을 이용한 분석

- 하이퍼 매개변수 C, gamma, kernel 의 최적값을 구하기 위하여 여러 C, gamma, kernel 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

SVC 모델 평가

```
======= SVC =======
 {'C': 0.5, 'gamma': 'auto', 'kernel': 'linear'}
=== Best Score ===
 0.8364729813389936
=== Train ===
 0.9149199610860529
=== Test ===
 0.8457729041386629

오차 행렬
[[1248  166]
 [ 270 1143]]
정확도: 0.8458, 정밀도: 0.8732, 재현율: 0.8089,    F1: 0.8398, AUC:0.8458
              precision    recall  f1-score   support

    negative       0.82      0.88      0.85      1414
    positive       0.87      0.81      0.84      1413

    accuracy                           0.85      2827
   macro avg       0.85      0.85      0.85      2827
weighted avg       0.85      0.85      0.85      2827
```

- 최적의 파라미터 C : 0.5, gamma : auto , kernel : linear 에서 훈련 점수 0.91 / 테스트 점수 0.84 확인됨

- AUC 정확도 84.5% / ROC_AUC 정확도 83.6%

Random Forest 모델

```
# RandomForest model
model = RandomForestClassifier(random_state=0)
params = {'n_estimators': [90, 100, 110],
          'max_depth': [40, 50, 60],
          'min_samples_leaf': [1, 2],
          'min_samples_split': [12, 14, 16]}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- Random Forest 모델을 이용한 분석

- 하이퍼 매개변수 n_estimators, max_depth, min_samples_leaf, min_samples_split의 최적값을 구하기 위하여 여러 n_estimators, max_depth, min_samples_leaf, min_samples_split 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

Random Forest 모델 평가

```
======= RandomForestClassifier =======
 {'max_depth': 60, 'min_samples_leaf': 2, 'min_samples_split': 6,
'n_estimators': 110}
=== Best Score ===
 0.7775714159370302
=== Train ===
 0.8071990802157956
=== Test ===
 0.7778563848602759

오차 행렬
[[1270  144]
 [ 484  929]]
정확도: 0.7779, 정밀도: 0.8658, 재현율: 0.6575,    F1: 0.7474, AUC:0.8667
              precision    recall  f1-score   support

    negative       0.72      0.90      0.80      1414
    positive       0.87      0.66      0.75      1413

    accuracy                           0.78      2827
   macro avg       0.79      0.78      0.77      2827
weighted avg       0.79      0.78      0.77      2827
```

- 최적의 파라미터 n_estimators : 110 , max_depth : 60 , min_samples_leaf : 2 , min_samples_split : 6에서 훈련 점수 0.80 / 테스트 점수 0.77 확인됨

- AUC 정확도 86.6% / ROC_AUC 정확도 77.7%

Decision Tree 모델

```
# DecisionTree model
model = DecisionTreeClassifier(random_state=0, criterion="entropy")
params = {'max_depth': [60, 100, 160],
          'min_samples_leaf': [1, 2],
          'min_samples_split': [12, 14, 20]}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- Decision Tree 모델을 이용한 분석

- 하이퍼 매개변수 max_depth, min_samples_leaf, min_samples_split 의 최적값을 구하기 위하여 여러 max_depth, min_samples_leaf, min_samples_split 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

Decision Tree 모델 평가

```
======= DecisionTreeClassifier =======
 {'max_depth': 160, 'min_samples_leaf': 1, 'min_samples_split': 20}
=== Best Score ===
 0.722384363668524
=== Train ===
 0.8647740337843813
=== Test ===
 0.7265652635302441

오차 행렬
[[1198  216]
 [ 557  856]]
정확도: 0.7266, 정밀도: 0.7985, 재현율: 0.6058,    F1: 0.6889, AUC:0.7766
              precision    recall  f1-score   support

    negative       0.68      0.85      0.76      1414
    positive       0.80      0.61      0.69      1413

    accuracy                           0.73      2827
   macro avg       0.74      0.73      0.72      2827
weighted avg       0.74      0.73      0.72      2827
```

- 최적의 파라미터 max_depth : 100 , min_samples_leaf : 1 , min_samples_split : 20에서 훈련 점수 0.86 / 테스트 점수 0.72 확인됨

- AUC 정확도 77.6% / ROC_AUC 정확도 72.2%

XGBoost 모델

```
# XGBoost model
model = xgb.XGBClassifier(random_state=0)
params = {'max_depth': [6, 8, 10],  # 3 ~ 10 적정
          'gamma': [0.5, 1, 2]}
estimator, score = model_training(
    model, params, x_train, x_test, y_train, y_test)
# 상위 모델 3개를 추출하기 위해 다른 df에 저장
temp = pd.DataFrame({'model': [estimator], 'score': [score]})
tr_model = tr_model.append(temp)
```

- XGBoost 모델을 이용한 분석

- 하이퍼 매개변수 max_depth, gamma의 최적값을 구하기 위하여 여러 max_depth, gamma 값을 params로 지정하여 model_training 함수에서 GridSearchCV 방식으로 최적화 모델 구현

XGBoost 모델 평가

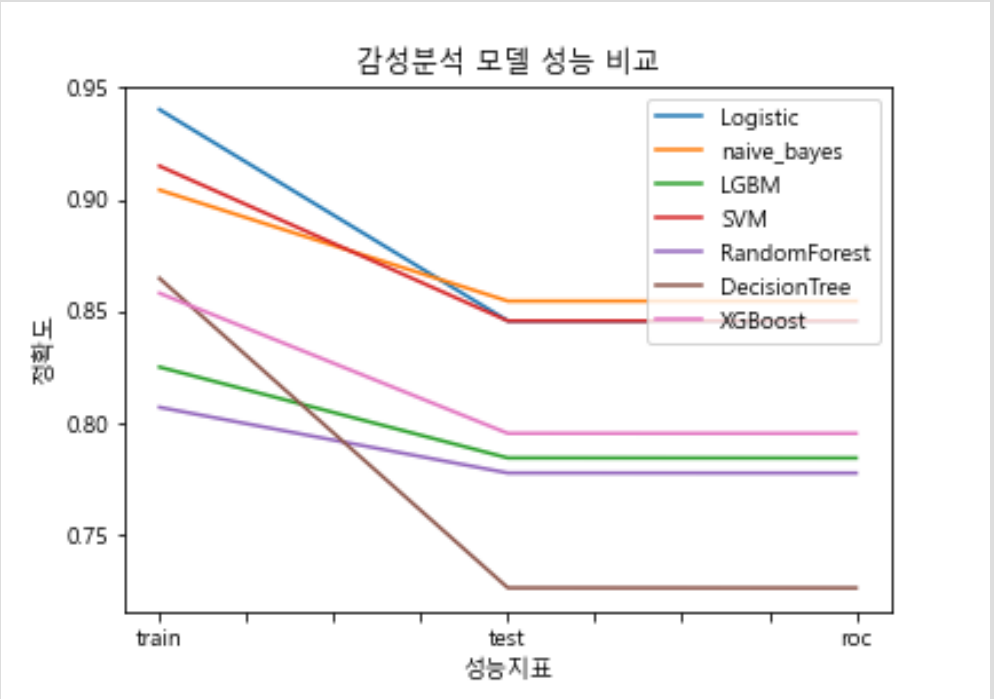
```
======= XGBClassifier =======
 {'gamma': 2, 'max_depth': 10}
=== Best Score ===
 0.7834969487927833
=== Train ===
 0.8581409746174936
=== Test ===
 0.7955429784223559

오차 행렬
[[1257  157]
 [ 421  992]]
정확도: 0.7955, 정밀도: 0.8634, 재현율: 0.7021,    F1: 0.7744, AUC:0.8847
              precision    recall  f1-score   support

    negative       0.75      0.89      0.81      1414
    positive       0.86      0.70      0.77      1413

    accuracy                           0.80      2827
   macro avg       0.81      0.80      0.79      2827
weighted avg       0.81      0.80      0.79      2827
```

- 최적의 파라미터 max_depth : 10 , gamma : 2에서 훈련 점수 0.85 / 테스트 점수 0.79
  확인됨

- AUC 정확도 88.4% / ROC_AUC 정확도 78.3%

모델 성능 시각화

```
# 모든 모델 평가 시각화
for idx in range(len(tr_model['model'])):
    train = tr_model['model'][idx].score(x_train, y_train)
    test = tr_model['model'][idx].score(x_test, y_test)

    predict = tr_model['model'][idx].predict(x_test)
    roc = roc_auc_score(y_test, predict)

    # model_df.iloc[:, idx] = [train, test, roc]
    model_df[model_df.columns[idx]] = [train, test, roc]

print(model_df)

# 좋은 모델을 판별하기 위해 시각화
plt.rc('font', family='Malgun Gothic')

model_df.plot()

plt.title('감성분석 모델 성능 비교')
plt.xlabel('성능지표')
plt.ylabel('정확도')
plt.savefig('MachineLearning_Performance_Indicator.png')
plt.pause(0.01)
# 가장 좋은 모델 3가지를 반환
best_models = tr_model.sort_values(by='score', ascending=False).head(3)

best_models = best_models.reset_index(drop = True)

return best_models, tfidf
```



- Matplotlib.pyplot 클래스를 이용하여 각 모델 별 train/test/roc score를 시각화
- LogisticRegression, Navie_Bayes, SVM 모델이 높은 성능을 보임

프로그램 실행 및 감성 예측 평가

머신러닝 및 텍스트 감성 예측

```python
if __name__ == '__main__':
    data = pd.read_csv('data/nlp.csv', encoding='utf-8')
    flag = [0, 0]
    flag[0] = 1
    # 모델 훈련 (많은 시간이 소요됩니다.)
    best_models, tfidf = search_best_estimator(data)

#%% main()
if __name__ == '__main__':
    main(best_models, tfidf, flag)
```

- 감성 분석/예측 진행 시 마다 data/nlp.csv의 토큰화 및 모델학습을 반복적으로 진행하는 것을 방지하기 위하여 학습 / 예측 분리

- 머신러닝/모델링 진행 후 정의된 best_models 및 tfidf객체를 매개변수로 하는 main 함수 실행

```
=== 시작화면 ===
1. 웹 데이터 크롤링
2. 모델 예측
3. 프로그램 종료

메뉴 선택:
```

- main 함수 실행 시 시작화면 출력됨

웹 데이터 크롤링

```python
import nvapi_module as nvapi
# 웹 데이터를 크롤링 하는 함수
def crawling_data():
    web_data_file_name = nvapi.webcrawler()

    web_data_json = pd.read_json(web_data_file_name, orient='str')

    web_data_json['document'] = web_data_json['description']

    web_data_json = data_cleansing(web_data_json)

    return web_data_json, web_data_file_name
```

```
=== 시작화면 ===
1. 웹 데이터 크롤링
2. 모델 예측
3. 프로그램 종료

메뉴 선택: 1

검색어를 입력하세요: 코로나
[2023-02-28 13:31:05.080236] Url Request Success
[2023-02-28 13:31:05.408338] Url Request Success
[2023-02-28 13:31:05.752065] Url Request Success
[2023-02-28 13:31:06.128531] Url Request Success
[2023-02-28 13:31:06.441025] Url Request Success
[2023-02-28 13:31:06.753489] Url Request Success
[2023-02-28 13:31:07.081607] Url Request Success
[2023-02-28 13:31:07.378459] Url Request Success
[2023-02-28 13:31:07.706560] Url Request Success
[2023-02-28 13:31:08.050272] Url Request Success
HTTP Error 400: Bad Request
[2023-02-28 13:31:08.112770] Error for URL : https://openapi.naver.com/v1/
search/news.json?query=%EC%BD%94%EB%A1%9C%EB
%82%98&start=1001&display=100&sort=sim
전체 검색 : 12071864 건
가져온 데이터 : 1000 건
코로나_naver_news.json SAVED
```

- 프로젝트에 제공된 nvapi.py를 모듈화하여 진행
- 웹 데이터 크롤링 메뉴 선택 시 실행되며, 크롤링 된 웹데이터의 description 컬럼에 대해 data_cleansing 함수가 실행됨
- 검색어_naver.news.json 파일 생성

best_models를 이용한 감성 예측

```python
def predict_data(best_models, text, size):
    best_models['predict'] = ''
    model_num = len(best_models)

    # 뉴스 데이터 예측
    for i in range(model_num):
        model = best_models['model'][i]
        best_models['predict'][i] = model.predict(text)

    # 3개의 모델끼리 예측값 비교
    predict_same = 0
    predict_different = 0
    for i in range(size):
        if (best_models['predict'][0][i] ==
            best_models['predict'][1][i] ==
            best_models['predict'][2][i]):
            predict_same += 1
        else:
            predict_different += 1

    # 모든 모델과 같은 값으로 예측하는 예측률
    print("모델들의 예측률")
    print(str(predict_same / (predict_same + predict_different) * 100) + "%")
```

- data_cleansing이 진행된
- 머신러닝/모델링 진행 후 정의된 best_models 및 tfidf객체를 매개변수로 하는 main 함수 실행
- main 함수 실행 시 시작화면 출력됨

```
=== 시작화면 ===
1. 웹 데이터 크롤링
2. 모델 예측
3. 프로그램 종료

메뉴 선택: 2
모델들의 예측률
83.89999999999999%
```

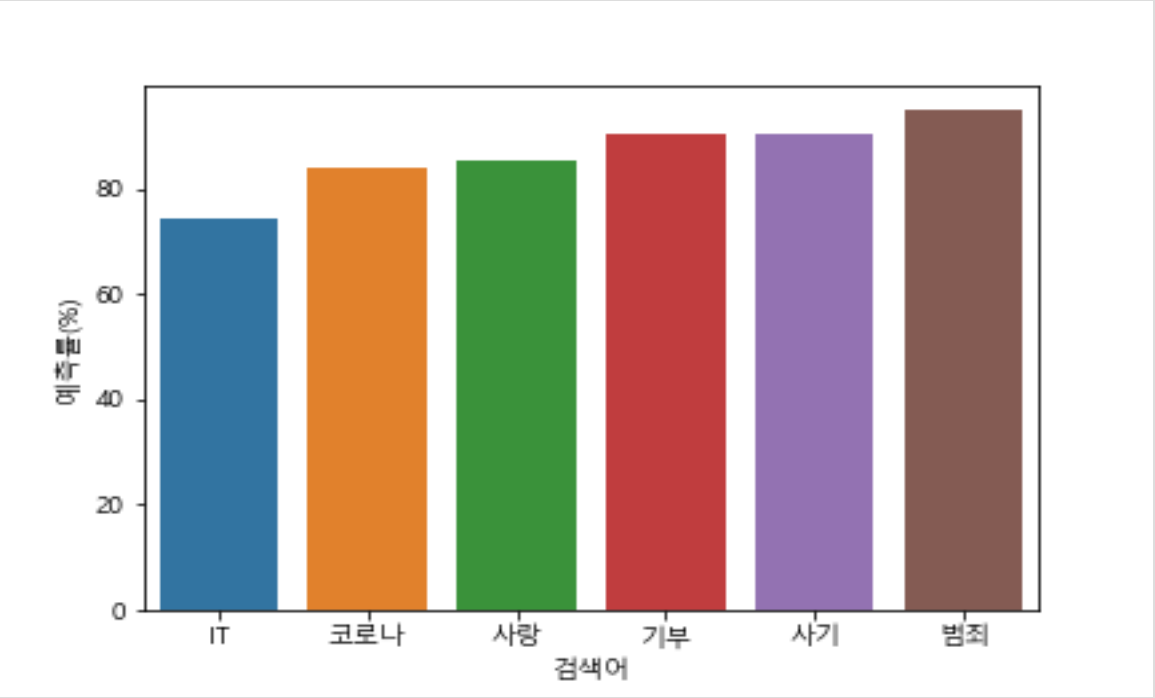
best_models를 이용한 감성 예측

```python
def predict_data(best_models, text, size):
    best_models['predict'] = ''
    model_num = len(best_models)

    # 뉴스 데이터 예측
    for i in range(model_num):
        model = best_models['model'][i]
        best_models['predict'][i] = model.predict(text)

    # 3개의 모델끼리 예측값 비교
    predict_same = 0
    predict_different = 0
    for i in range(size):
        if (best_models['predict'][0][i] ==
            best_models['predict'][1][i] ==
            best_models['predict'][2][i]):
            predict_same += 1
        else:
            predict_different += 1

    # 모든 모델과 같은 값으로 예측하는 예측률
    print("모델들의 예측률")
    print(str(predict_same / (predict_same + predict_different) * 100) + "%")
```

```
=== 시작화면 ===
1. 웹 데이터 크롤링
2. 모델 예측
3. 프로그램 종료

메뉴 선택: 2
모델들의 예측률
83.89999999999999%
```

- data_cleansing이 진행된 웹데이터를 best_models로 예측하는 함수
- 세 개의 best_model로 웹데이터의 description 컬럼을 예측함
- 위 예측값을 토대로 세 개의 모델의 예측값이 일치하는지 예측률을 계산하여 출력

프로그램의 감성 예측률 평가



- 중립(IT, 코로나), 긍정(사랑, 기부), 부정(사기, 범죄) 총 6개의 키워드로 프로그램을 실행하여 예측률을 비교한 결과이다.
- 부정적인 단어일수록 예측률이 높으며, 긍정적인 단어, 중립적인 단어 순으로 예측률이 감소하는것을 확인 할 수 있다.
- 현재 모델이 부정적인 쪽에 대한 정확도는 어느 정도 높으나 긍정적인 부분에 대해서는 부정적인 부분보다는 약간 낮은 것을 알 수 있다.