머신러닝 프로젝트

: 영화 리뷰 데이터를 활용한 감성분석

BDA X 이지스 스터디 — 7조

김인서 37
박지영 24
박지수 75
박혜원 91
이주미 48

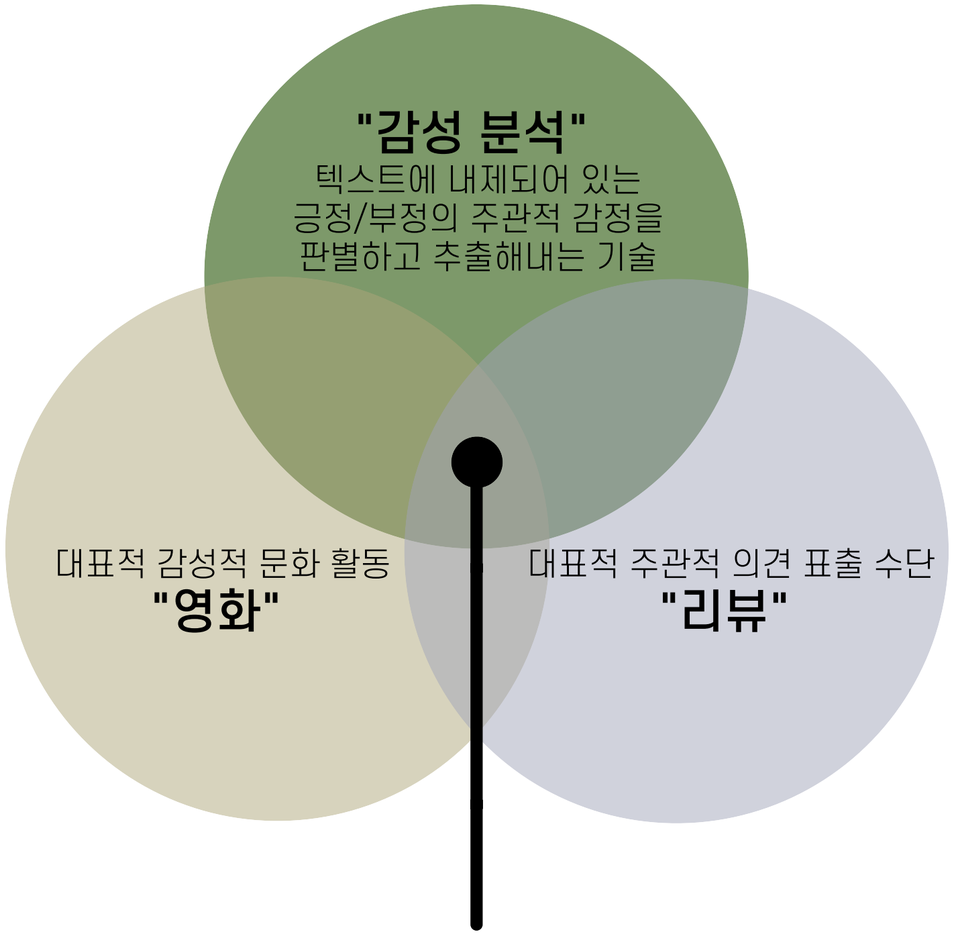
CONTENTS INDEX

프로젝트 개요

01

주제 선정 배경
프로젝트 목표
프로젝트 과정

주제 선정 배경



"영화 리뷰 감성 분석"

이라는 감성 분석 분야 중 가장 특징적이며 대표적인 주제 선정

프로젝트 목표

토큰화

2 글자 이상의 명사, 형용사, 동사만 추출

1 글자 포함이며 모든 형태소를 추출

X

분석 모델

- LSTM
- Logistic Regression
- SVM

총 6가지의 결과 도출 후
각 모델 결과값 비교

프로젝트 과정

01

데이터 수집

영화에 대한 리뷰어들의 긍정/부정 판단이 담긴 텍스트 리뷰 데이터셋 수집

02

데이터 정제

중복값, NA값, 불용어 제거 등 데이터 정제 과정

03

토큰화

2글자 이상의 명사, 형용사, 동사만 추출하는 방식과 모든 형태소를 추출하는 두 가지의 토큰화 방식 설계

04

벡터화 / 패딩

각 분석 모델에 사용되기 알맞게 벡터화 또는 패딩을 적절히 적용하는 과정

05

데이터 분석

LSTM, Logistic Regression, SVM의 세 가지 모델을 통한 분석

06

결론 도출

적용한 토큰화별, 모델별 결과 비교

데이터 수집

02

데이터셋

데이터셋

활용한 데이터셋

Naver sentiment movie corpus v1.0
(https://github.com/e9t/nsmc)

네이버 영화 리뷰 데이터에 대한 감성 분류가
라벨링 된 텍스트 데이터셋

데이터셋 구조

데이터 구조	id	document	label
내용	리뷰 작성자 정보	영화 리뷰 댓글	영화에 대한 평가 (긍정 1, 부정 0)

데이터 전처리

03

데이터 정제
불용어 제거
토큰화
벡터화 / 패딩

데이터 정제

결측치/ 중복값 처리

결측치, 중복값 확인

```
# 각 data의 결측치 확인
df_train.info()
print()
df_test.info()
```

```
# 각 data의 중복값 확인
tr_dup = sum(df_train.duplicated())
te_dup  = sum(df_test.duplicated())
```

결측치, 중복값 제거

```
# train과 test에 있는 결측치와 중복을 모두 제거

df_train.drop_duplicates(inplace=True)
df_train.dropna(inplace=True)

df_test.drop_duplicates(inplace=True)
df_test.dropna(inplace=True)
```

index 리셋

```
# 제거 결과 index가 연속적이지 않으므로 index를 리셋

df_train.reset_index(drop=True, inplace=True)
df_test.reset_index(drop=True, inplace=True)
```

불필요한 문자 처리

한글, 공백 제외한
나머지 제거

```
# 한글, 공백을 제외한 나머지는 모두 제거
# 오타로 인해 맞춤법이 틀리는 경우 형태소 분석이 불가능하므로 공백 유지

df_train_1['document'] = df_train_1['document'].apply(lambda x: re.sub("[^ㄱ-ㅎㅏ-ㅣ가-힣\s]", "", x))
df_test_1['document'] = df_test_1['document'].apply(lambda x: re.sub("[^ㄱ-ㅎㅏ-ㅣ가-힣\s]", "", x))
```

문자열 0인 경우 제거

```
# 문자열 길이가 0이 되는 경우를 제거

for idx in range(len(df_train_1)):
    if df_train_1.loc[idx, 'document'] == '' or df_train_1.loc[idx, 'document'].isspace():
        df_train_1.drop(idx, axis=0, inplace=True)

for idx in range(len(df_test_1)):
    if df_test_1.loc[idx, 'document'] == '' or df_test_1.loc[idx, 'document'].isspace():
        df_test_1.drop(idx, axis=0, inplace=True)
```

index 리셋

```
# 마찬가지로 index를 0부터 연속적으로 만들기 위해 리셋

df_train_1.reset_index(drop=True, inplace=True)
df_test_1.reset_index(drop=True, inplace=True)
```

데이터 정제

정제 전 후 데이터 비교

	document	label
0	아 더빙.. 진짜 짜증나네요 목소리	0
1	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	너무재밋었다그래서보는것을추천한다	0
3	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1

test 데이터 정제 결과

	document	label
0	아 더빙 진짜 짜증나네요 목소리	0
1	흠포스터보고 초딩영화줄오버연기조차 가볍지 않구나	1
2	너무재밋었다그래서보는것을추천한다	0
3	교도소 이야기구먼 솔직히 재미는 없다평점 조정	0
4	사이몬페그의 익살스런 연기가 돋보였던 영화스파이더맨에서 늙어보이기만 했던 커스틴 던...	1

	document	label
0	굳 ㅋ	1
1	GDNTOPCLASSINTHECLUB	0
2	뭐야 이 평점들은.... 나쁘진 않지만 10점 짜리는 더더욱 아니잖아	0
3	지루하지는 않은데 완전 막장임... 돈주고 보기에는....	0
4	3D만 아니었어도 별 다섯 개 줬을텐데.. 왜 3D로 나와서 제 심기를 불편하게 하죠??	0

train 데이터 정제 결과

	document	label
0	굳 ㅋ	1
1	뭐야 이 평점들은 나쁘진 않지만 점 짜리는 더더욱 아니잖아	0
2	지루하지는 않은데 완전 막장임 돈주고 보기에는	0
3	만 아니었어도 별 다섯 개 줬을텐데 왜 로 나와서 제 심기를 불편하게 하죠	0
4	음악이 주가 된 최고의 음악영화	1

불용어 제거

긍정/부정 리뷰 별로 출현 빈도가 높은 단어를 분류했을 때,
긍정/부정 모두에 자주 나타나는 단어는 판별력이 없으므로
동시에 자주 나타나는 단어를 "불용어"로 판단

긍정/부정 리뷰 별로 사용된 명사 분류

```
# 명사만 뽑아서 긍정 리뷰인 경우에는 긍정 리스트에, 부정 리뷰에는 부정 리스트에 추가

nouns_positive = []
nouns_negative = []

tags_noun = ['NNG', 'NNP', 'NNB', 'NP', 'NR']

for idx in range(len(df_train_1)):
    tokens = komoran.pos(df_train_1.loc[idx, 'document'])
    for token in tokens:
        if token[1] in tags_noun:
            if df_train_1.loc[idx, 'label'] == 1:
                nouns_positive.append(token[0])
            elif df_train_1.loc[idx, 'label'] == 0:
                nouns_negative.append(token[0])
    if idx % 10000 == 0: # 진행 상황 출력을 위한 출력 코드
        print(idx)
```

긍정/부정 리뷰 리스트 속 명사의 출현 빈도 계산

```
# CountVectorizer를 이용해 단어의 출현 빈도를 계산한다.

vectorizer_pos = CountVectorizer()
vectorizer_neg = CountVectorizer()

vec_positive = vectorizer_pos.fit_transform(nouns_positive)
vec_negative = vectorizer_neg.fit_transform(nouns_negative)

word_pos_list = vectorizer_pos.get_feature_names_out()
count_pos_list = vec_positive.toarray().sum(axis=0)

word_neg_list = vectorizer_neg.get_feature_names_out()
count_neg_list = vec_negative.toarray().sum(axis=0)
```

불용어 제거

```
[('영화', 27250),
 ('최고', 5452),
 ('연기', 3783),
 ('감동', 3593),
 ('생각', 3285),
 ('드라마', 2902),
 ('평점', 2610),
 ('사람', 2439),
 ('사랑', 2251),
 ('배우', 2078)]
```

명사 - 긍정 리뷰

```
[('재밌', 6603),
 ('재미있', 3224),
 ('만들', 2697),
 ('나오', 2571),
 ('느끼', 1625),
 ('이렇', 1385),
 ('모르', 1358),
 ('보이', 1283),
 ('아름답', 1265),
 ('좋아하', 1169)]
```

동사 & 형용사 - 긍정 리뷰

```
[('정말', 5837),
 ('너무', 5623),
 ('어요', 5594),
 ('는데', 5556),
 ('네요', 4941),
 ('지만', 4535),
 ('으로', 4414),
 ('에서', 4249),
 ('ㅂ니다', 4154),
 ('습니다', 3887)]
```

나머지 - 긍정 리뷰

```
[('영화', 26188),
 ('평점', 3490),
 ('스토리', 3424),
 ('연기', 3246),
 ('쓰레기', 3143),
 ('내용', 2888),
 ('감독', 2775),
 ('시간', 2657),
 ('사람', 2447),
 ('배우', 2273)]
```

명사 - 부정 리뷰

```
[('만들', 4777),
 ('나오', 3856),
 ('아깝', 3846),
 ('재미없', 3024),
 ('이렇', 2070),
 ('모르', 1581),
 ('보이', 1449),
 ('그렇', 1446),
 ('재밌', 1314),
 ('좋아하', 790)]
```

동사 & 형용사 - 부정 리뷰

```
[('는데', 5667),
 ('너무', 4972),
 ('으로', 4675),
 ('아니', 4462),
 ('ㄴ다', 4234),
 ('아서', 4006),
 ('에서', 3737),
 ('진짜', 3422),
 ('이런', 3047),
 ('지만', 2977)]
```

나머지 - 부정 리뷰

```
stop_words_noun     = ['영화', '연기', '평점', '사람', '배우']
stop_words_verb_adj = ['만들', '나오', '이렇', '모르', '보이']
stop_words_etc      = ['너무', '는데', '지만', '으로', '에서']
stop_words = stop_words_noun + stop_words_verb_adj + stop_words_etc
```

▲ 최종 불용어

토큰화

토큰화 1

2 글자 이상의 명사, 형용사, 동사만 추출

```
for idx in range(len(df_train_1)):
    tmp = []
    tokens = komoran.pos(df_train_1.loc[idx, 'document'])
    for token in tokens:
        if token[1] in tags_include and len(token[0]) > 1 and token[0] not in stop_words:
        # 명사, 동사, 형용사만 고려, 2글자 이상만 고려, 불용어 제거
            tmp.append(token[0])
    if len(tmp) > 0:
        train_tokens_1.append(tmp)
        train_target_1.append(df_train_1.loc[idx, 'label'])
    if idx % 10000 == 0: # 진행 상황 확인
        print(idx)
```

토큰화 2

1 글자 포함이며 모든 형태소를 추출

```
for idx in range(len(df_train_1)):
    tmp = []
    tokens = komoran.pos(df_train_1.loc[idx, 'document'])
    for token in tokens:
        if token[0] not in stop_words:
        # 모든 형태소 고려, 불용어 제거
            tmp.append(token[0])
    if len(tmp) > 0:
        train_tokens_2.append(tmp)
        train_target_2.append(df_train_1.loc[idx, 'label'])
    if idx % 10000 == 0: # 진행 상황 확인
        print(idx)
```

벡터화 / 패딩

벡터화

Logistic Regression, SVM에 활용
=> TF-IDF로 중요도 가중치 부여

```
vectorizer_tfidf_1st = TfidfVectorizer(min_df=5, analyzer=lambda x: x)
vectorizer_tfidf_2nd = TfidfVectorizer(min_df=5, analyzer=lambda x: x)

train_vectors_tfidf_1st = vectorizer_tfidf_1st.fit_transform(train_tokens_1st)
test_vectors_tfidf_1st = vectorizer_tfidf_1st.transform(test_tokens_1st)

train_vectors_tfidf_2nd = vectorizer_tfidf_2nd.fit_transform(train_tokens_2nd)
test_vectors_tfidf_2nd = vectorizer_tfidf_2nd.transform(test_tokens_2nd)
```

패딩

LSTM 모델에 활용
=> 길이 조절

```
test_1st_sequences = tokenizer_1st.texts_to_sequences(test_tokens_1st)
test_1st_padded = pad_sequences(test_1st_sequences, maxlen=max_len_1st)

test_2nd_sequences = tokenizer_2nd.texts_to_sequences(test_tokens_2nd)
test_2nd_padded = pad_sequences(test_2nd_sequences, maxlen=max_len_2nd)
```

데이터 분석

04

각 모델에 대한 이해
LSTM
Logistic Regression
SVM

각 모델에 대한 이해

LSTM

- RNN의 한 종류
- RNN unit에 이전 데이터의 정보를 저장하고 있는 메모리 셀 함수와, 메모리 셀을 유지할 것인지 업데이트 할 것인지 결정하는 게이트 함수를 추가한 것
- 감성분석에 사용되는 many to one 모델 사용해 많은 입력을 넣었을 때 하나의 출력을 내는 구조 반영

Logistic Reg

- 이진분류 문제에 사용되는 머신러닝 모델
- 로지스틱 함수를 사용하여 확률을 예측하고, 이를 기반으로 분류함
- 모델 결과를 확률로 제공해 해석에 용이하며, 피쳐의 스케일에 덜 민감
- 다만 비선형 문제에는 다소 비적합하고 이상치에 민감하며, 고차원 데이터에서는 성능이 제한됨

SVM

- 분류와 회귀에 모두 적용할 수 있는 지도 학습 알고리즘
- 훈련 데이터셋에서 두 클래스 간의 최대 margin을 찾는 것을 목표로 하며, 이 최대 margin을 가지는 결정 경계를 찾는 방식으로 동작
- 선형과 비선형으로 구분되나, 본 프로젝트에서는 학습 시간을 고려하여 선형 SVM을 이용

LSTM - 분석

모델 세팅

Embedding output_dim = 100
LSTM units = 128
dropout = 0.2
recurrent_dropout = 0.2

```
model_1st = tf.keras.Sequential()
model_1st.add(tf.keras.layers.Embedding(28373, 100, input_shape=(None,)))
model_1st.add(tf.keras.layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model_1st.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

모델에 checkpoint와 earlystopping 도입으로 성능 향상

```
# 각각의 batch_size에 대해 val_loss가 최소인 모델만 저장
checkpoint_1st_32 = tf.keras.callbacks.ModelCheckpoint(filepath='/content/drive/
                                                       monitor='val_loss',
                                                       verbose=1,
                                                       save_best_only=True)

# 3번 epoch를 반복했음에도 새로운 최솟값이 나오지 않으면 정지.
earlystopping_1st = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                     patience=3,
                                                     verbose=1)
```

LSTM - 분석

batch = 32인 case의 분석

```
history = model_1st.fit(train_1st_padded,
                        train_target_1st,
                        epochs=10,
                        callbacks=[earlystopping_1st, checkpoint_1st_32],
                        batch_size=32,
                        validation_split=0.2,
                        verbose=1)
```

```
Epoch 1/10
3368/3368 [==============================] - ETA: 0s - loss: 0.4704 - accuracy: 0.7702
Epoch 1: val_loss improved from inf to 0.44476, saving model to /content/drive/MyDrive/Data Analysis/BDA 7기/Easy Study Project/프로젝트/
3368/3368 [==============================] - 252s 73ms/step - loss: 0.4704 - accuracy: 0.7702 - val_loss: 0.4448 - val_accuracy: 0.7894
Epoch 2/10
3368/3368 [==============================] - ETA: 0s - loss: 0.3996 - accuracy: 0.8163
Epoch 2: val_loss did not improve from 0.44476
3368/3368 [==============================] - 246s 73ms/step - loss: 0.3996 - accuracy: 0.8163 - val_loss: 0.4457 - val_accuracy: 0.7896
Epoch 3/10
3368/3368 [==============================] - ETA: 0s - loss: 0.3527 - accuracy: 0.8404
Epoch 3: val_loss did not improve from 0.44476
3368/3368 [==============================] - 247s 73ms/step - loss: 0.3527 - accuracy: 0.8404 - val_loss: 0.4698 - val_accuracy: 0.7856
Epoch 4/10
3368/3368 [==============================] - ETA: 0s - loss: 0.3156 - accuracy: 0.8584
Epoch 4: val_loss did not improve from 0.44476
3368/3368 [==============================] - 245s 73ms/step - loss: 0.3156 - accuracy: 0.8584 - val_loss: 0.5034 - val_accuracy: 0.7824
Epoch 4: early stopping
```

```
predict = loaded_model_1st.evaluate(test_1st_padded,
                                    test_target_1st,
                                    verbose=1)
print(predict)
```

```
1414/1414 [==============================] - 19s 14ms/step - loss: 0.4436 - accuracy: 0.7912
[0.4436452090740204, 0.7912430167198181]
```

LSTM - 분석 결과

batch=32	loss	accuracy	예측시간
토큰화 1	0.4436	0.7912	19s 14ms/step
토큰화 2	0.3543	0.8457	31s 20ms/step

batch=64	loss	accuracy	예측시간
토큰화 1	0.4443	0.7895	20s 14ms/step
토큰화 2	0.3589	0.8426	30s 19ms/step

batch=128	loss	accuracy	예측시간
토큰화 1	0.4450	0.7904	19s 13ms/step
토큰화 2	0.3645	0.8406	31s 20ms/step

batch=256	loss	accuracy	예측시간
토큰화 1	0.4460	0.7889	19s 13ms/step
토큰화 2	0.3672	0.8386	30s 19ms/step

LSTM - 분석 결과

토큰화 1	loss	accuracy	예측시간
batch=32	0.4436	0.7912	19s 14ms/step
batch=64	0.4443	0.7895	20s 14ms/step
batch=128	0.4450	0.7904	19s 13ms/step
batch=256	0.4460	0.7889	19s 13ms/step

토큰화 2	loss	accuracy	예측시간
batch=32	0.3543	0.8457	31s 20ms/step
batch=64	0.3589	0.8426	30s 19ms/step
batch=128	0.3645	0.8406	31s 20ms/step
batch=256	0.3672	0.8386	30s 19ms/step

Logistic Regression - 분석

최적의 값을 찾기 위해
파라미터 튜닝(GridSearchCV) 적용

1. solver (최적화 알고리즘)
- 'liblinear' : 데이터셋이 작고 이진 분류 문제에 주로 사용
- 'lbfgs','sag' : 데이터셋이 크고 다중 분류 문제에 주로 사용

2. max_iter (최대 반복 횟수)
- 모델이 수렴하지 않을 경우 max_iter 값을 늘려 해결 가능성 높임
- 너무 크게 설정하면 모델 훈련 시간이 길어질 수 있다는 단점
- 가장 보편적으로 많이 사용하는 수치인 기본 값 100부터 점차 늘리는 방향으로 설정

2. params (정규화 매개변수 C)
- 모델의 규제 강도를 조절
- 작은 C 값은 강한 규제를, 큰 C 값은 약한 규제를 의미
- 대부분의 경우, 0.1에서 10 사이의 범위에서 튜닝됨

모델 세팅 및 분석

```
lr = LogisticRegression(random_state=0)

params = {'C': [0.1, 1, 10],
          'solver': ['liblinear', 'lbfgs', 'sag'],
          'max_iter': [100, 500, 1000]}

grid_cv = GridSearchCV(lr, param_grid=params, cv=3, scoring='accuracy', verbose=2)
grid_cv.fit(X_train, y_train)

print(grid_cv.best_params_, round(grid_cv.best_score_,3))
```

```
# 최적의 parameter값과 최고의 score 점수
grid_cv.best_params_, grid_cv.best_score_
```

```
({'C': 1, 'max_iter': 100, 'solver': 'lbfgs'}, 0.7861368573347306)
```

```
best_lr = LogisticRegression(random_state=0, solver='liblinear', max_iter=1000, C=grid_cv.best_params_['C'])
best_lr.fit(X_train, y_train)

y_pred = best_lr.predict(X_test)

accuracy_1 = accuracy_score(y_test, y_pred)
f1_1 = f1_score(y_test, y_pred)

print("Accuracy:", round(accuracy_1, 3))
print("F1 Score:", round(f1_1, 3))
```

```
Accuracy: 0.791
F1 Score: 0.788
```

Logistic Regression - 분석 결과

	accuracy	F1 scrore
토큰화 1	0.791	0.788
토큰화 2	0.833	0.831

SVM - 분석

선형 커널 SVM 분류 모델인
LinearSVC 적용

1. loss (default='squared_hinge')
- loss function 적용 방식을 설정
- 'hinge'는 standard한 SVM loss이며, 'squared_hinge'는 hinge loss를 제곱한 것
- 'squared_hinge'를 적용하면 큰 loss에 대해 더 강한 penalty 부여

2. C (default=1.0)
- 규제 강도의 역수
- C가 작으면 모형이 단순해져 결정 경계가 부드러워짐 (오차를 어느 정도 허용하는 소프트 마진)
- C가 크면 더 많은 학습 데이터를 정확히 분류하도록 동작(오차를 최대한 허용하지 않으려 하는 하드 마진)
- C가 너무 적으면 너무 많은 오차를 허용하게 되므로 과소적합을, C가 너무 크면 과대적합을 유발

모델 세팅 및 분석

```
param_grid = {
    'C': [0.01, 0.05, 0.1, 0.5, 1, 5, 10],
    'loss': ['hinge', 'squared_hinge'],
}

grid_search_1st = GridSearchCV(svc_1st, param_grid, cv=5, scoring='f1', verbose=3)
grid_search_2nd = GridSearchCV(svc_2nd, param_grid, cv=5, scoring='f1', verbose=3)
```

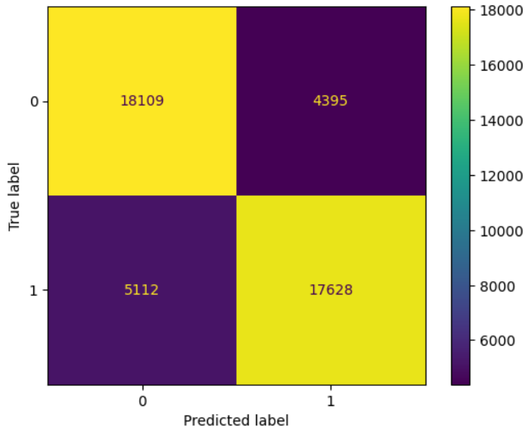
```
grid_search_1st.fit(train_tfidf_1st, train_target_1st)
```

```
grid_search_1st.best_params_, grid_search_1st.best_score_
```

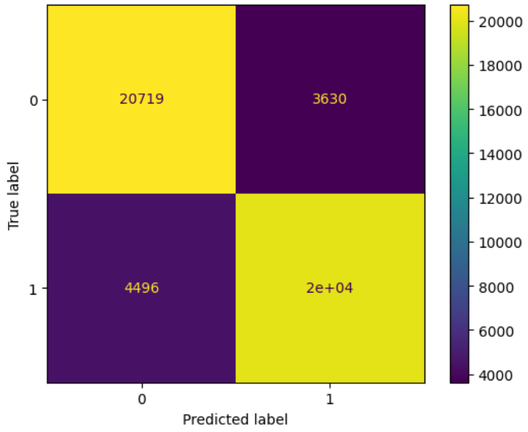
```
({'C': 0.1, 'loss': 'squared_hinge'}, 0.7823090671929631)
```

SVM - 분석 결과

	accuracy	F1 score	Precision	Recall	ROC-AUC
토큰화 1	0.790	0.788	0.800	0.775	0.790
토큰화 2	0.834	0.831	0.847	0.817	0.834



토큰화 1
ConfusionMatrixDisplay



토큰화 2
ConfusionMatrixDisplay

결론

05

각 토큰화 별 분석 결과 비교
각 모델 별 분석 결과 비교

각 토큰화 별 분석 결과 비교

토큰화 1	accuracy	F1 score
LSTM	0.7912	
Logistic Regression	0.791	0.788
SVM	0.790	0.788

토큰화 2	accuracy	F1 score
LSTM	0.8457	
Logistic Regression	0.833	0.831
SVM	0.834	0.831

일부 형태소만 반영한 토큰화1보다,
모든 형태소를 반영한 토큰화2가
높은 정확도를 보이는 유의미한 결과

각 모델 별 분석 결과 비교

LSTM	accuracy
토큰화 1	0.7912
토큰화 2	0.8457

Logistric Regression	accuracy	F1 score
토큰화 1	0.791	0.788
토큰화 2	0.833	0.831

SVM	accuracy	F1 score
토큰화 1	0.790	0.788
토큰화 2	0.834	0.831

모델 별 분석 결과는 큰 차이가 없으나,
LSTM이 다른 모델에 비해
비교적 다소 높은 정확도를 보임

THANKYOU

BDA x 이지스 스터디

7조