기계학습 감성 분석 기말 과제

학번 : 201921030

이름 : 임재석

목차

1-1

감성분석이란?

감성 분석(Sentiment Analysis)이란 텍스트에 들어있는 의견이나 감성, 평가, 태도 등의 주관적인 정보를 컴퓨터를 통해 분석하는 과정

응용

감성 분석은 마케팅에서 고객 서비스, 임상 의학에 이르기까지 다양한 애플리케이션을 위한 리뷰 및 설문 조사 응답, 온라인 및 소셜 미디어, 의료 자료 등 고객의 소리 자료에 널리 적용

1-2

전처리

모듈 호출 및 세팅

```
%pip install pandas numpy scikit-learn nltk matplotlib seaborn

import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_curve, auc
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import nltk
import matplotlib.pyplot as plt
import seaborn as sns

# NLTK 리소스 다운로드
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

우선 ML에 필요한 모듈 호출과 감성 분석에 필요한 NLTK 소스를 호출

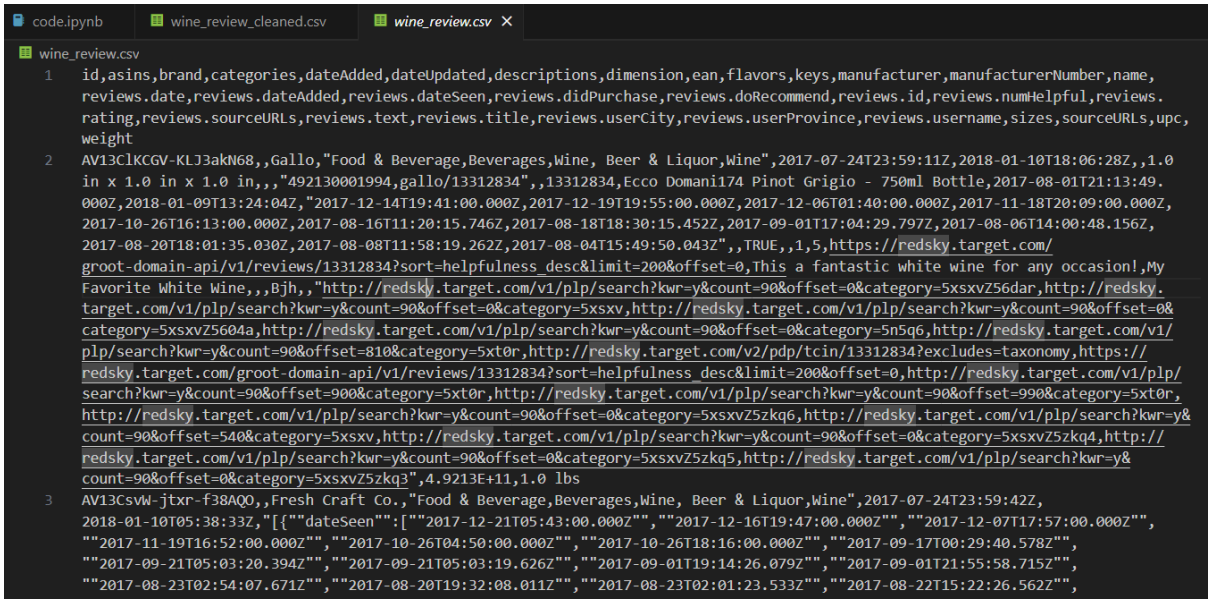
```python
# 데이터 불러오기
df = pd.read_csv('./wine_review.csv')

# 필요한 열만 선택
df = df[['reviews.text', 'reviews.rating']]

# 결측값 제거
df.dropna(subset=['reviews.text', 'reviews.rating'], inplace=True)

# 감성 라벨링 (5점 기준 4, 5는 긍정, 1, 2는 부정, 3은 중립으로 간주)
def label_sentiment(rating):
    if rating >= 4:
        return 'positive'
    elif rating <= 2:
        return 'negative'
    else:
        return 'neutral'

df['sentiment'] = df['reviews.rating'].apply(label_sentiment)
```

불러온 데이터를 호출하고 기초적인 결측치 제거와 라벨링을 진행

```python
# 텍스트 전처리 함수 정의
def preprocess_text(text):
    # 소문자 변환
    text = text.lower()
    # HTML 태그 제거
    text = re.sub(r'<.*?>', '', text)
    # 특수 문자 및 숫자 제거
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # 토큰화
    words = word_tokenize(text)
    # 불용어 제거
    words = [word for word in words if word not in stopwords.words('english')]
    # 표제어 추출
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)

# 텍스트 데이터 전처리 적용
df['cleaned_text'] = df['reviews.text'].apply(preprocess_text)

# 데이터셋 분할
X = df['cleaned_text']
y = df['sentiment']

# 저장할 전처리된 데이터를 새 CSV 파일로 저장
df.to_csv('wine_review_cleaned.csv', index=False)

print("Data preprocessing and CSV file creation completed.")
```

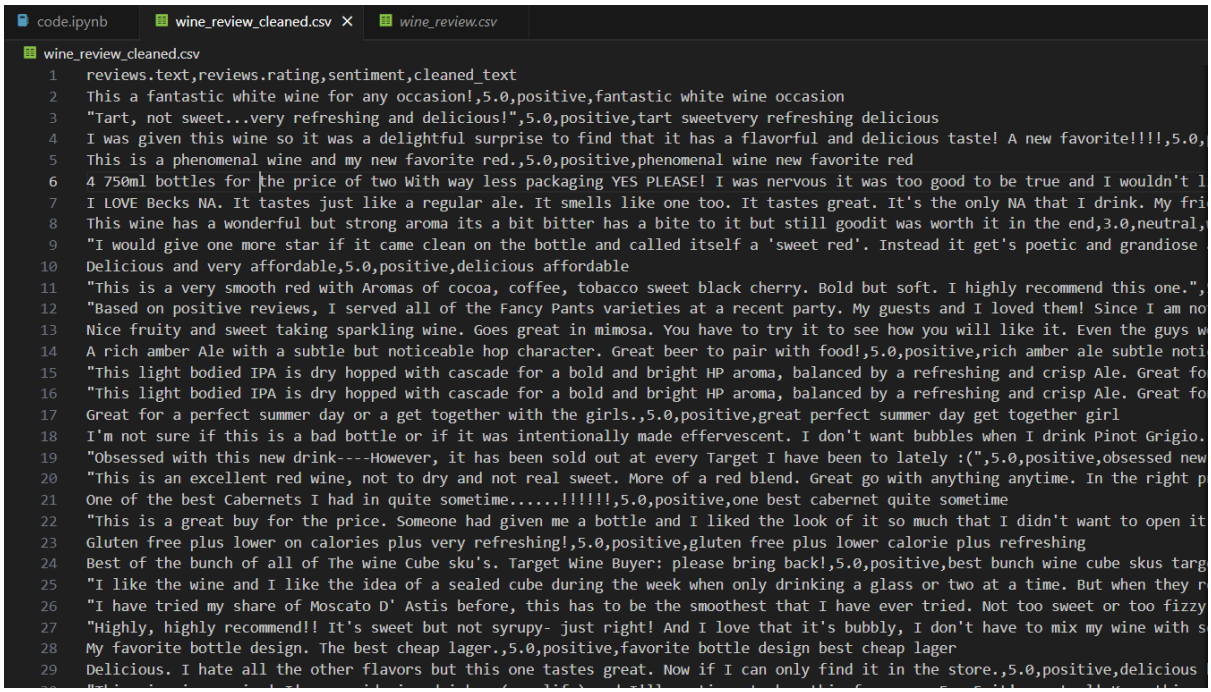
요구사항에 맞춘 토큰화와 각 문자들을 전처리하고 새로운 데이터셋을 csv로 저장

BEFORE Data Set



전처리 전 데이터는 각 컬럼과 필요없는 날짜, 링크 등 데이터가 존재

AFTER Data Set



토큰 제거와 필요한 평가, 감성 텍스트만 남은 데이터셋을 확인가능

3 -1 모델

모델 생성 및 튜닝

```
# 데이터 불러오기 (전처리된 데이터 사용)
df = pd.read_csv('./wine_review_cleaned.csv')

# 결측값 처리
df.dropna(subset=['cleaned_text'], inplace=True)

# 데이터셋 분할
X = df['cleaned_text']
y = df['sentiment']

# 학습 및 테스트 세트로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# 파이프라인 설정
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('svc', SVC())
])
```

위 코드를 통해 전처리를 거친 데이터를 호출하고 학습을 위해 분할.

분할은 다음과 같음

X = df['cleaned_text']: 텍스트 데이터를 특징 변수

y = df['sentiment']: 감성 데이터를 목표 변수

train_test_split(X, y, test_size=0.2, random_state=42, stratify=y): 데이터를 학습 세트와 테스트 세트로 80:20 비율로 분할 stratify=y는 원본 데이터의 클래스 비율을 유지하기 위해 사용

Pipeline: 여러 처리 단계를 묶어주는 역할

SVC(): 서포트 벡터 머신 분류기를 사용

TfidfVectorizer(): 텍스트 데이터를 TF-IDF(Term Frequency-Inverse Document Frequency) 벡터로 변환

하이퍼파라미터 튜닝

```
# 하이퍼파라미터 튜닝
param_grid = {
    'tfidf__max_df': [0.8, 0.9, 1.0],
    'tfidf__ngram_range': [(1, 1), (1, 2)],
    'svc__C': [0.1, 1, 10],
    'svc__kernel': ['linear', 'rbf']
}

# GridSearchCV를 사용하여 최적의 하이퍼파라미터 탐색
grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1, scoring='accuracy')
grid_search.fit(X_train, y_train)

# 최적의 모델로 예측 수행
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# 모델 평가
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))

# 모델 평가 (교차 검증)
cross_val_scores = cross_val_score(best_model, X, y, cv=5, scoring='accuracy')
print(f"Cross-validation scores: {cross_val_scores}")
print(f"Mean cross-validation score: {cross_val_scores.mean()}")
```

GridSearchCV에서 사용할 하이퍼파라미터 그리드 사용

GridSearchCV: 교차 검증을 통해 최적의 하이퍼파라미터를 탐색

grid_search.best_params_: 최적의 하이퍼파라미터를 출력

accuracy_score(y_test, y_pred): 테스트 세트의 정확도를 계산

classification_report(y_test, y_pred): 정밀도, 재현율, F1 점수를 포함한 상세한 분류 보고서를 출력

cross_val_score(best_model, X, y, cv=5, scoring='accuracy'): 전체 데이터에 대해 5-폴드 교차 검증을 수행하여 정확도를 평가

cross_val_scores.mean(): 교차 검증의 평균 정확도를 계산

튜닝 결과

```
Best Parameters: {'svc__C': 10, 'svc__kernel': 'rbf', 'tfidf__max_df': 0.8, 'tfidf__ngram_range': (1, 1)}
Accuracy: 0.934560327198364
              precision    recall  f1-score   support

    negative       0.75      0.14      0.23        22
     neutral       0.00      0.00      0.00        13
    positive       0.94      1.00      0.97       454

    accuracy                           0.93       489
   macro avg       0.56      0.38      0.40       489
weighted avg       0.90      0.93      0.91       489

/home/codespace/.local/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1517: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/codespace/.local/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1517: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/codespace/.local/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1517: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Cross-validation scores: [0.93047035 0.92229039 0.92842536 0.93237705 0.93237705]
Mean cross-validation score: 0.9291880384860371
```

최적의 하이퍼파라미터

svc__C: 10: SVM 분류기의 규제 매개변수 CCC가 10으로 설정 높은 CCC 값은 모델이 데이터에 더 잘 맞게 되지만, 과적합(overfitting)의 위험 가능

- **svc__kernel: 'rbf'**: RBF (Radial Basis Function) 커널을 사용하여 비선형 경계를 학습
- **tfidf__max_df: 0.8**: TF-IDF 벡터라이저에서 단어가 전체 문서의 80% 이하에서 나타나면 포함
- **tfidf__ngram_range: (1, 1)**: Unigram (단일 단어)를 사용

테스트 세트에서 93.46%의 높은 정확도를 달성

negative 클래스:

- **precision: 0.75**: 모델이 negative 클래스로 예측한 것 중 75%가 실제로 negative
- **recall: 0.14**: 실제 negative 샘플 중 14%만이 정확히 예측
- **f1-score: 0.23**: 정밀도와 재현율의 조화 평균을 나타냄. 낮은 값은 모델이 negative 클래스에서 잘 작동하지 않음을 나타냈음

neutral 클래스:

- **precision: 0.00**, **recall: 0.00**, **f1-score: 0.00**: 모델이 neutral 클래스를 전혀 예측하지 못함. 이는 neutral 샘플이 너무 적어서 모델이 이 클래스를 제대로 학습하지 못한 결과를 도출함

positive 클래스:

- **precision: 0.94**: 모델이 positive 클래스로 예측한 것 중 94%가 실제로 positive
- **recall: 1.00**: 실제 positive 샘플 중 100%가 정확히 예측
- **f1-score: 0.97**: 매우 높은 값으로, 모델이 positive 클래스에서 매우 잘 작동하는 것을 알 수있음

전체 정확도: 0.93으로, 모델이 전반적으로 높은 정확도를 달성

macro avg: 각 클래스의 평균 성능을 나타내며, 이 값이 낮은 이유는 neutral 클래스의 성능이 매우 저조함

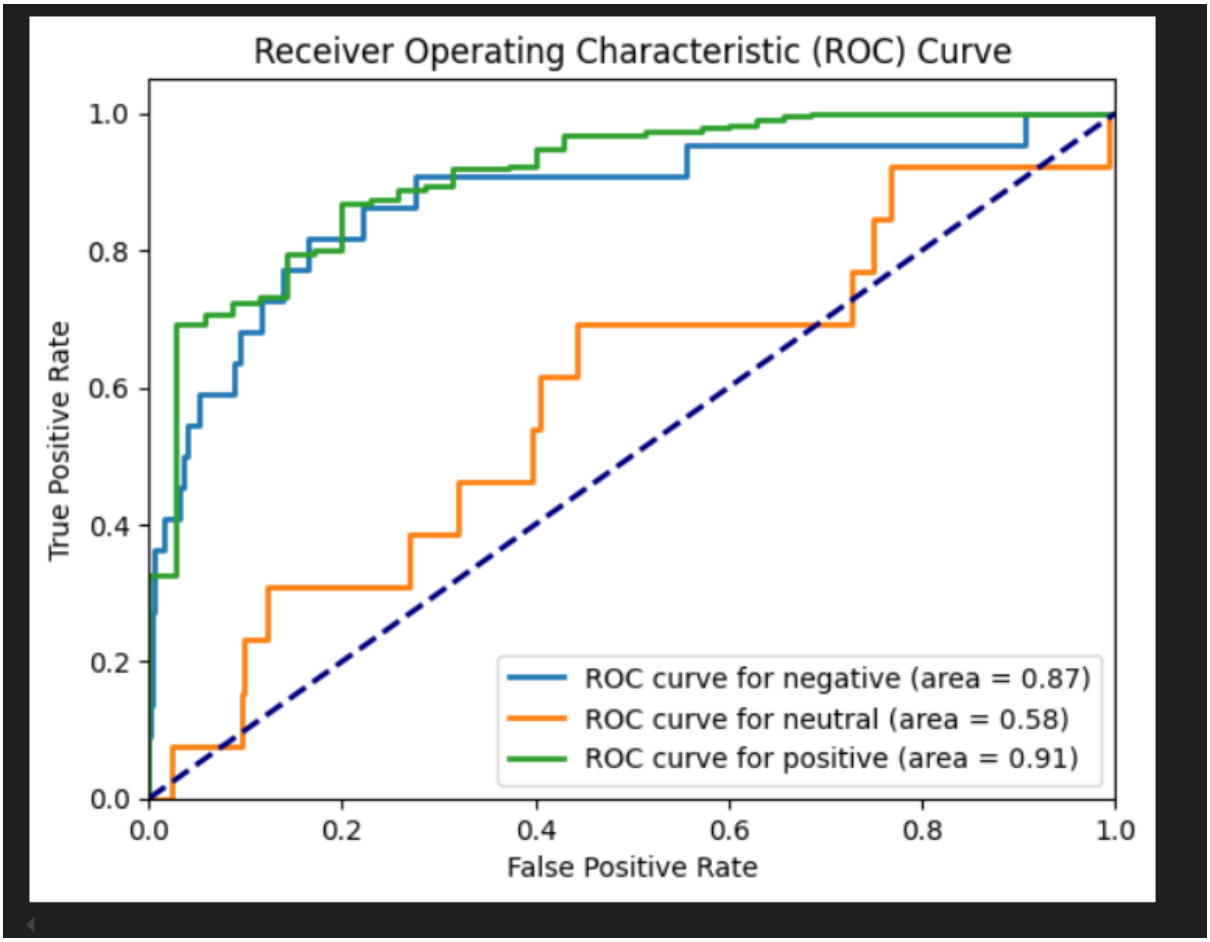
weighted avg: 클래스의 지원 수를 고려한 가중 평균입니다. 전체적인 성능을 더 잘 나타냅니다.

교차 검증 정확도: 각 폴드에서 높은 정확도를 유지하고 있으며, 평균 교차 검증 정확도가 약 92.92%입니다. 이는 모델이 일관되게 잘 작동함을 나타냄

튜닝 결론

- 모델은 positive 클래스에서 매우 잘 작동하며, 전반적인 정확도도 높음
- negative와 neutral 클래스의 성능은 낮으며, 특히 neutral 클래스는 전혀 예측되지 않았음
- 데이터 불균형 문제가 있으며, 이를 해결하기 위해 데이터 증강이나 클래스 가중치를 조정하는 방법을 고려해야함
- 모델의 전반적인 성능은 우수하지만, 모든 클래스에서 균형 잡힌 성능을 얻기 위해 추가적인 조정이 필요하다 판단됨

5 시각화



분석

각 클래스별 ROC 곡선 및 AUC 값

부정 클래스 (Negative)

ROC 곡선: 파란색
AUC: 0.87

분석: AUC 값이 0.87로 높아 부정 클래스에 대한 모델의 성능이 좋음을 나타냄

중립 클래스 (Neutral)

ROC 곡선: 주황색
AUC: 0.58

분석: AUC 값이 0.58로, 중립 클래스에 대한 모델의 성능이 낮음. 이 클래스에 대한 예측 성능이 저조함을 알수있음.

긍정 클래스 (Positive)

ROC 곡선: 초록색
AUC: 0.91

분석: AUC 값이 0.91로 매우 높아 긍정 클래스에 대한 모델의 성능이 매우 좋음을 나타냅니다.

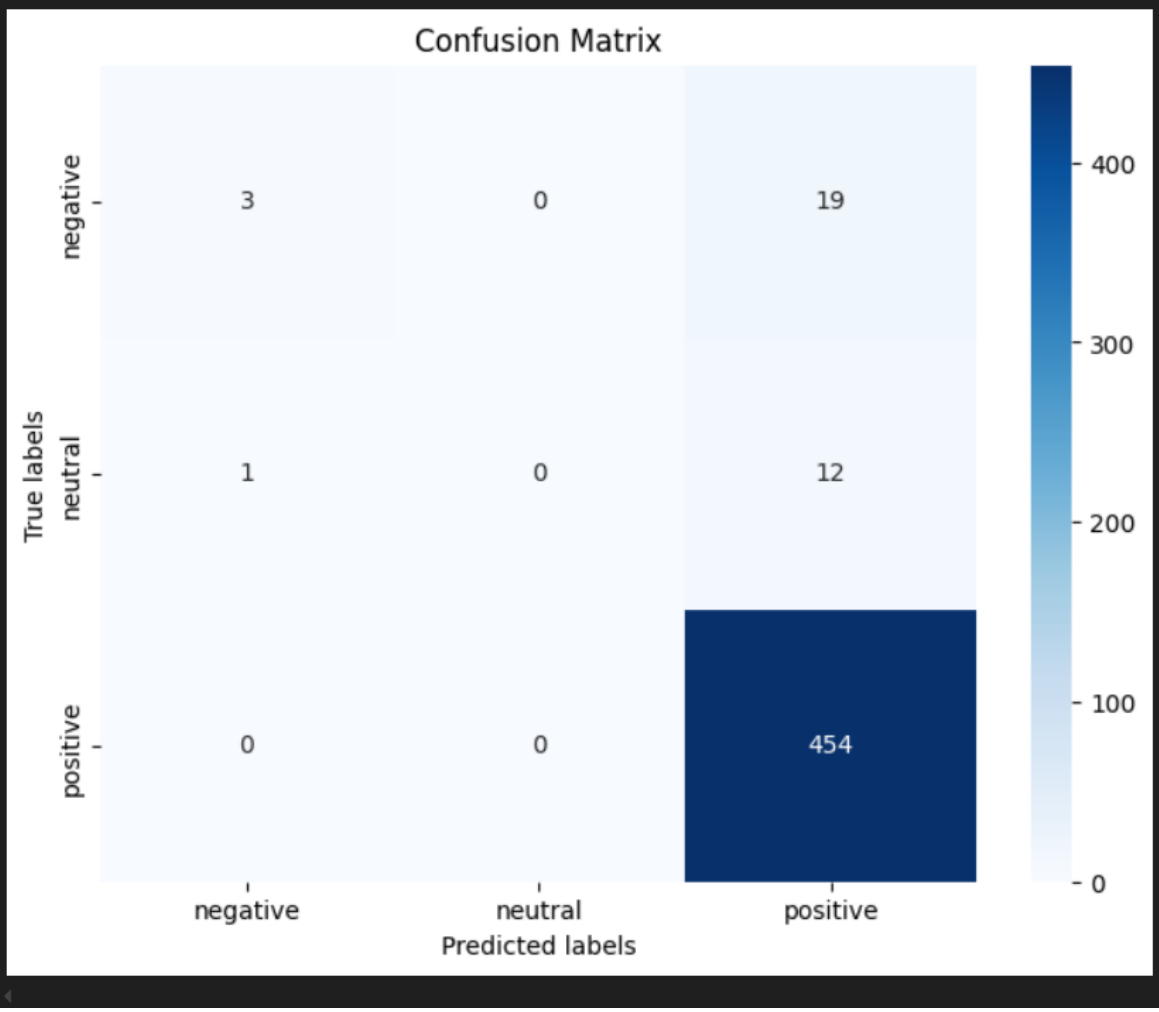
요약 및 결론

긍정 클래스는 AUC 값이 0.91로 가장 높아, 이 클래스에 대한 예측 성능이 매우 뛰어남

부정 클래스도 AUC 값이 0.87로 높아, 모델이 부정 클래스를 비교적 잘 예측하고 있음

중립 클래스는 AUC 값이 0.58로, 모델이 중립 클래스를 잘 예측하지 못지만. 이는 앞서 혼동행렬 분석에서 확인한 바와 같이 중립 클래스를 거의 긍정 클래스로 예측하는 경향과 일치

혼돈 행렬 분석



Negative (부정) 클래스:

총 22개 (3 + 0 + 19) 중 실제 부정인 22개 데이터 중 3개는 부정으로 정확하게 예측되었고, 19개는 긍정으로 잘못 예측.
단, 중립으로 예측된 경우는 없음

Neutral (중립) 클래스:

총 13개 (1 + 0 + 12) 중 실제 중립인 13개 데이터 중 1개는 부정으로, 12개는 긍정으로 잘못 예측.
중립으로 예측된 경우는 없음

Positive (긍정) 클래스:

총 454개 (0 + 0 + 454) 중 실제 긍정인 454개 데이터는 모두 긍정으로 정확하게 예측.