기계학습(8585)

기말고사 대체 과제

소프트웨어전공

202020899

박보겸

<목차>

I. 감성분석의 기본 이해

감성 분석이란 디지털 텍스트를 분석한 후 메시지의 감정적 어조가 긍정적인지, 부정적인지 또는 중립인지를 확인하는 프로세스이다. 이러한 감성분석을 이용해 고객의 리뷰를 객관적으로 평가하여 결과를 제공한다. 사람이 직접 리뷰를 평가하면 긍정적인 리뷰인지 부정적인 리뷰인지 두 단어를 모두 사용했거나 사용한 단어가 애매한 경우 주관적으로 판단해버리지만, 인공지능을 통해 감정을 분류하면 객관적인 방식으로 텍스트를 정렬, 분류하게 되므로 정확도가 올라간다. 또한 긍정, 부정 두 가지의 감정을 모두 반영할 수 있다. 이러한 방식으로 브랜드나 소셜 미디어를 모니터링하고, 시장 조사를 하고, 브랜드 평가를 올리는 등의 목적을 갖고 사용하고 있다.

감성분석 모델의 예로는 규칙 기반 감성 분석 도구인 VADER, 전통 머신 러닝 모델인 SVM, Logistic Regression, 딥러닝 기반 모델인 RNN, LSTM, CNN등이 있다.

이를 응용하여 감성 분석을 진행할 것인데, 신발 리뷰에 관한 자료인 'amazon_uk_shoes_products_dataset_2021_12.csv'를 보면, 'review_text'라는 필드에 사용자가 리뷰를 직접 작성한 내용이 적혀 있다. 이 텍스트를 감성 분석하여 사용자가 숫자로 평가를 나타낸 수치인 'review_rating'과 얼마나 일치하는지 확인할 것이다.

II. 데이터 전처리

머신 러닝 작업을 하기 전 데이터의 품질을 높이고 분석 결과의 신뢰성을 보장하기 위해 데이터 전처리를 진행했다.

```
df['review_text'] = df['review_text'].fillna('') # 결측값 제거
df['review_text'] = df['review_text'].astype(str) # 문자열 변환

df['review_text'] = df['review_text'].apply(lambda x: re.sub(r'[^₩w₩s]', '', str(x)))   # 특수 문자 제거
df['review_text'] = df['review_text'].apply(lambda x: re.sub(r'₩d+', '', x))               # 숫자 제거
df['review_text'] = df['review_text'].apply(lambda x: re.sub(r'₩s+', ' ', x).strip()) # 불필요한 공백 제거

stop_words = set(stopwords.words('english')) # 불용어 리스트 변수에 저장
df['review_text'] = df['review_text'].apply(lambda x: ' '.join([word for word in word_tokenize(x) if word.lower() not in stop_words]))

stemmer = PorterStemmer()  # 스테밍 적용
df['review_text'] = df['review_text'].apply(lambda x: ' '.join([stemmer.stem(word) for word in word_tokenize(x)]))

lemmatizer = WordNetLemmatizer()# 표제어 추출 적용
df['review_text'] = df['review_text'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in word_tokenize(x)]))
```

데이터셋에서 누락된 값을 처리하여 분석의 정확도를 높이도록 결측 값을 제거하고, 데이터 형식을 통일하여 분석의 일관성을 확보하기 위해 문자열을 변환한다.

데이터에서 불필요한 정보를 제거하여 분석의 정확성, 신뢰성을 높이기 위해 노이즈 제거를 실행한다. (특수 문자 제거, 숫자 제거, 불필요한 공백 제거)

분석의 효율성을 높이기 위해 'review_text'를 토큰화 한 후 불용어에 해당하지 않는 단어만 포함시키는 리스트를 생성하였다.

일관된 데이터 분석, 모델 성능 향상을 위해 스테밍과 표제어 추출을 하였다.

III. 모델 선택 및 학습

수업에서 활용한 모델 중 VADER을 사용하였다. NLTK라이브러리를 사용해 VADER감성분석기를 불러와 텍스트의 감정을 쉽게 접근, 사용할 수 있다.

```
vader_sentiment = SentimentIntensityAnalyzer()
def calc_sentiment(review): # 감정 점수 계산 함수
    return vader_sentiment.polarity_scores(review)["compound"] # VADER 감정 분석의 compound 점수
df['sentiment_score'] = df['review_text'].apply(calc_sentiment) # 감정 점수 컬럼 추가
```

	review_text	sentiment_score
0	love look convers half price uniqu ive never s...	0.9188
1	shoe cute nd day wear tongu start rip rd day w...	0.2240
2	good qualiti	0.4404
3	great	0.6249
4	ho scelto il modello bianco con rifinitura die...	-0.4588

먼저, VADER 감정 분석기를 초기화한 후 감정 점수 계산 함수를 작성하여 계산하였다. 이때 VADER 감정 분석의 compound 점수는 -1~1 사이의 값을 가지며, 감정의 농도를 나타내기 때문에 다음과 같은 결과 값이 나타난다. 결과를 보면 가장 첫번째 값처럼 긍정적인 리뷰는 1에 가까운 약 0.92의 수치, 5번째 리뷰처럼 부정적인 리뷰는 -1에 가까운 -0.5의 수치를 보이는 것을 알 수 있다.

추가로 감정 분석 모델 VADER이 제대로 분석했는지 판단하고 싶어 'review_rating'값과 비교해보았다.

```
def check_labeling(review_rating, sentiment_label): # 리뷰의 별점과 라벨링된 감정 점수가 일치하는지 확인하는 함수
    if review_rating >= 2 and sentiment_label == 1:
        return "Match"
    elif review_rating < 2 and sentiment_label == 0:
        return "Match"
    else:
        return "Mismatch"

df['sentiment_label'] = df['sentiment_score'].apply(lambda x: 1 if x >= 0 else 0) # review_rating 과 라벨링된 감정 점수 비교
df['Check_Labeling'] = df.apply(lambda row: check_labeling(row['review_rating'], row['sentiment_label']), axis=1)

# Match 로 출력되는 비율 계산
match_count = (df['Check_Labeling'] == 'Match').sum() # 'Check_Labeling' 열에서 'Match'인 항목의 수를 세고 match_count 에 저장
total_count = len(df) # DataFrame 의 총 행 수를 total_count 에 저장
match_ratio = match_count / total_count # 'Match'로 출력되는 비율을 계산하여 match_ratio 에 저장
```

```
      review_rating  sentiment_label Check_Labeling
0               5.0                1          Match
1               2.0                1          Match
2               5.0                1          Match
3               5.0                1          Match
4               5.0                0       Mismatch
...             ...              ...            ...
6818            5.0                1          Match
6819            5.0                1          Match
6820            5.0                1          Match
6821            5.0                0       Mismatch
6822            5.0                0       Mismatch

[6823 rows x 3 columns]
matching ratio 0.8330646343250769
```

주요 코드를 보자면, 긍정, 부정으로 나뉘도록 라벨링을 하면서 기존 csv파일 필드의 'review_rating' 점수와 얼마나 일치하는지 확인하는 함수를 작성하여 비교 후 Match, Mismatch로 출력되게 하였다. 이때, 5점 만점에 2점 이하는 부정적인 리뷰라고 생각하여 그 점수를 기준으로 계산하였고, 리뷰의 양이 많기 때문에 수치로 계산해야 정확한 판단이 될 것 같아 Match 항목의 수를 세어 적중률도 출력해보았다. 결과를 보면, 위에서 분석하였을 때 5번째 리뷰가 수치상 부정적인 리뷰로 판단되었는데 리뷰 점수는 5.0으로 되어있기 때문에 Mismatch로 출력이 된다. 이렇게 정확하지 않은 값들도 나타남을 알 수 있었고, 적중률이 약 0.83 정도로 출력된 것을 보아 높은 일치율을 보여주는 것을 알 수 있었다.

모델의 성능 향상을 위해 모델학습 및 튜닝을 추가적으로 진행하였다.

```
X = df['sentiment_score'].values.reshape(-1, 1) # 감정 점수`
y = df['sentiment_label'] # 감정 점수에 기반한 라벨 (위에서 라벨링한 것)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression() # 로지스틱 회귀 모델 초기화
param_grid = {'C': [0.1, 1, 10, 100]} # 하이퍼파라미터 그리드 정의
grid_search = GridSearchCV(model, param_grid, cv=5) # 그리드 서치 수행
grid_search.fit(X_train, y_train) # 학습 세트로 모델 학습

print("Best parameters found: ", grid_search.best_params_)

best_model = grid_search.best_estimator_ # 최적 모델을 best_model 에 할당
y_pred = best_model.predict(X_test) # X_text 데이터에 대한 예측값을 변수에 저장
```

먼저 특성(감정 점수)과 라벨을 분리하여 변수에 저장한 후, 이를 이용하여 모델을 학습시킨 다음 성능을 평가하기 위해 데이터셋 학습 세트와 테스트 세트로 분할하였다. 이때, 주어진 데이터셋을 학습 데이터와 테스트 데이터로 8:2 비율로 나누고, random_state 를 42 로 설정하여 데이터를 분할할 때 일관된 결과를 얻도록 하였다. 그 후 로지스틱 회귀 모델을 초기화하고, 하이퍼파라미터 그리드를 정의한 다음 그리드 서치를 수행하였다. 이때 cv=5 는 5 겹 교차 검증을 의미하는데 기본값이 5 라 5 로 지정하였다. 학습한 세트로 모델 학습을 진행하여 최종적으로 그리드 서치를 통해 최적의 하이퍼파라미터를 찾았다. 결과값으로 'Best parameters found: {'C': 100}'이 출력되어, 이 최적 모델을 이용하여 예측 값을 수행하여 변수에 저장하였다.

```
from sklearn.model_selection import cross_val_score

# 5-겹 교차 검증 수행을 통해 오버피팅 해결함

cv_accuracy = cross_val_score(best_model, X, y, cv=5, scoring='accuracy').mean() # 정확도
cv_precision = cross_val_score(best_model, X, y, cv=5, scoring='precision').mean() # 정밀도
cv_recall = cross_val_score(best_model, X, y, cv=5, scoring='recall').mean() # 재현률
cv_f1 = cross_val_score(best_model, X, y, cv=5, scoring='f1').mean() # F1 점수
cv_roc_auc = cross_val_score(best_model, X, y, cv=5, scoring='roc_auc').mean() # ROC-AUC

print("CV Accuracy: ", cv_accuracy)
print("CV Precision: ", cv_precision)
print("CV Recall: ", cv_recall)
print("CV F1 Score: ", cv_f1)
print("CV ROC-AUC Score: ", cv_roc_auc)
```

```
CV Accuracy:  0.9956035362486976
CV Precision:  0.9949772255876 51
CV Recall:  1.0
CV F1 Score:  0.9974808925241803
CV ROC-AUC Score:  1.0
```

오버피팅은 머신 러닝에서 모델이 훈련 데이터에 지나치게 적합하여 새로운 데이터에 대한 예측 성능이 저하되는 현상이다. 오버피팅된 모델의 경우 훈련 데이터에서 높은 성능을 보이더라도 테스트 데이터 혹은 실전에서 사용했을 때 낮은 성능을 보이기 때문에 이를 방지하기 위해 5-겹 교차 검증을 수행하여 오버피팅 문제를 해결하고자 했다. 검증을 통해 정확도, 정밀도, 재현율, F1점수, ROC-AUC 값을 출력해보았는데, 모든 수치가 100%에 가깝게 나타나 모델이 일반적인 데이터에서도 효과적으로 작동한다는 것을 알 수 있었다.

물론 모든 수치가 100%가 아니다. VADER모델은 단어 수준의 감정 점수를 사용하고, 초기 개발 당시 소셜 미디어 피드를 중심으로 훈련 및 평가되었기 때문에 완전히 문맥을 이해하지 못하였거나, 단순히 나누기 힘든 복합적인 감정의 단어가 포함되어 있는 등 약점이 있을 수 있으며 그로 인해 100%의 정확성을 보이기는 힘들 것이다.

IV. 평가

```
accuracy = accuracy_score(y_test, y_pred) # 정확도
precision = precision_score(y_test, y_pred) # 정밀도
recall = recall_score(y_test, y_pred) # 재현률
f1 = f1_score(y_test, y_pred) # F1 점수
roc_auc = roc_auc_score(y_test, y_pred) # ROC-AUC 점수

print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 Score: ", f1)
print("ROC-AUC: ", roc_auc)
```

```
Accuracy:  0.9963369963369964
Precision:  0.9957841483979764
Recall:  1.0
F1 Score:  0.9978876214617659
ROC-AUC:  0.9864130434782609
```

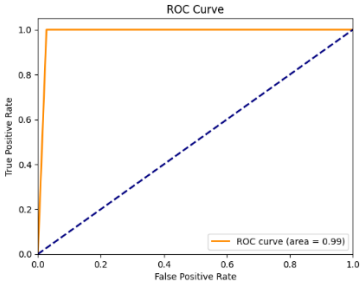
위에서 5-겹 교차 검증을 수행했을 때와 같은 방식으로 모델 평가를 수행하였다. 전체 예측 중 올바르게 예측한 비율인 정확도는 약 99.6%로 높은 정확도를 보였는데 이는 약 99.6%가 올바르게 분류되었다는 것을 의미한다. 두번째로, 모델이 긍정으로 예측한 것 중 실제로 긍정인 비율인 정밀도 역시 99.6%로 높은 수치를 보인다. 세번째로, 재현율도 비슷한 의미인데 모델이 실제로 긍정인 것 중 100%를 올바르게 예측했다는 것을 알 수 있다. 네번째로, F1 점수는 정밀도와 재현율의 조화 평균인데, 약 99.8%의 점수를 보였다. 마지막으로 ROC-AUC 점수는 ROC 곡선 아래 면적을 나타내며, 모델의 분류 성능을 시각화 하는데 사용된다. 출력 결과 약 98.6%의 높은 점수를 보이는 것을 보아 모델의 분류 성능이 매우 우수함을 알 수 있다.

오버피팅은 위에서 5-겹 교차 검증을 통해 성능을 평가하였을 때 우수함을 알 수 있었기 때문에 문제가 적다고 판단하였고, 정확도, 정밀도, 재현율, F1점수, ROC-AUC 값을 출력한 결과 좋은 성능이라는 결과가 출력되었기 때문에 훈련데이터, 테스트 데이터가 모두 낮은 성능을 보일 때 해결해야 하는 문제인 언더피팅 역시 문제가 적다고 판단하여 최종적으로 오버피팅과 언더피팅을 추가로 해결해야 할 필요가 없다고 판단하였다.

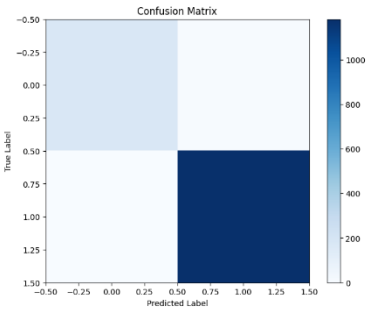
V. 시각화 및 해석

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, roc_curve, auc

y_pred = best_model.predict(X_test) # 예측 생성

confusion = confusion_matrix(y_test, y_pred) # 혼동 행렬 생성

fpr, tpr, _ = roc_curve(y_test, y_pred)  # ROC 곡선의 거짓 양성 비율 및 참 양성 비율 계산
roc_auc = auc(fpr, tpr)  # ROC 곡선 아래 영역(AUC) 계산

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)  # ROC 곡선 그리기
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # 대각선 기준선 그리기
plt.xlim([0.0, 1.0])  # X 축 범위 설정
plt.ylim([0.0, 1.05])  # Y 축 범위 설정
plt.xlabel('False Positive Rate')  # X 축 레이블 설정
plt.ylabel('True Positive Rate')  # Y 축 레이블 설정
plt.title('ROC Curve')  # 그래프 제목 설정
plt.legend(loc='lower right')  # 범례 위치 설정
plt.show()  # 그래프 출력

plt.figure(figsize=(8, 6))  # 그림 크기 설정
plt.imshow(confusion, cmap='Blues', interpolation='nearest')  # 혼동 행렬 그리기
plt.colorbar()  # 컬러 바 추가
plt.title('Confusion Matrix')  # 그래프 제목 설정
plt.xlabel('Predicted Label')  # X 축 레이블 설정
plt.ylabel('True Label')  # Y 축 레이블 설정
plt.show()  # 그래프 출력
```

모델의 진단 성능을 평가하는 그래프인 ROC 곡선과 모델 예측 성능을 실제 값과 비교하여 시각화한 혼동 행렬을 생성하여 모델 결과의 시각화를 진행하였다. 처음에는 먼저, matplot을 이용하여 시각화 해보았는데, 먼저 예측을 생성하고 혼동 행렬과 ROC곡선을 생성한 뒤 곡선과 행렬을 그려 시각화를 진행하였다.



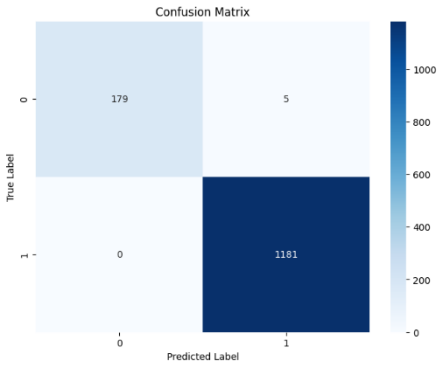
해당 그림을 보면, true positive rate와 false positive rate의 관계를 나타내는데, 이때 주황색 선은 모델의 성능, 대각선의 파란색 점선은 랜덤 추측의 성능을 나타낸다. 주황색 선, 즉 roc곡선은 왼쪽 위 모서리에 가까울수록 성능이 우수함을 나타내고, 이 곡선이 대각선의 파란색 점선보다 위에 있을 때 모델의 성능이 좋음을 나타낸다.



혼동행렬 구역은 왼쪽위부터 TP(실제 양성인 샘플을 양성으로 올바르게 예측한 경우), FN(실제 양성인 샘플을 음성으로 잘못 예측한 경우), FP(실제 음성인 샘플을 양성으로 잘못 예측한 경우), TN(실제 음성인 샘플을 음성으로 올바르게 예측한 경우)로 구역이 나뉘는데 숫자가 표시되지는 않았지만, 음영의 진하기로 각 범주의 크기를 시각화한 결과를 보면, 진한 색상의 영역이 주로 TP,

TN쪽에 위치한 것을 보여 양성, 음성 샘플을 올바르게 예측했음을 나타낸다.

```
import seaborn as sns

# 혼동 행렬 시각화
plt.figure(figsize=(8, 6))  # 그림 크기 설정
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues")  # 혼동 행렬 그리기
plt.title('Confusion Matrix')  # 그래프 제목 설정
plt.xlabel('Predicted Label')  # X 축 레이블 설정
plt.ylabel('True Label')  # Y 축 레이블 설정
plt.show()  # 그래프 출력
```



roc곡선의 결과값은 matplot, seaborn 모두 일치하지만, matplot에서는 숫자를 직접 추가하지 않으면 표시가 되지 않아 아쉬웠는데 seaborn을 사용하면 셀 안에 숫자를 기본적으로 표시하기 때문에 추가로 혼동 행렬 시각화를 진행하였다. 수치를 보면, tn: 179, fp:5, fn:0, tp:1181로 양성으로 예측한 샘플 중 실제 양성인 비율과 음성으로 예측한 샘플 중 실제 음성인 비율이 매우 높은 것을 볼 수 있었다.

따라서 세 개의 시각화 된 그림을 통해 구현한 모델의 성능이 매우 좋음을 알 수 있다.