보고서

**0. 감성분석이란?**

감성 분석은 텍스트 데이터에서 사람들의 감정, 의견, 태도 등을 자동으로 식별하고 추출하는 과정으로, 자연어 처리(NLP)와 텍스트 분석의 한 분야입니다. 리뷰, 소셜 미디어 포스트, 뉴스 기사 등의 텍스트 데이터를 분석하여 긍정, 부정, 중립과 같은 감정 범주로 분류하는 작업을 포함합니다. 감성 분석은 마케팅, 고객 서비스, 브랜드 모니터링, 금융, 정치 등 다양한 산업 분야에서 활용됩니다. 마케팅에서는 소비자 리뷰와 소셜 미디어 피드백을 분석하여 고객의 감정을 파악하고, 이를 바탕으로 마케팅 전략을 수립하거나 경쟁사의 제품에 대한 소비자 반응을 분석하여 시장 동향을 파악합니다. 고객 서비스에서는 고객의 문의나 불만 사항을 자동으로 분석하여 품질을 향상시키고, 실시간으로 고객의 감정을 모니터링하여 즉각적인 대응을 통해 만족도를 높입니다. 브랜드 모니터링에서는 소셜 미디어와 리뷰 사이트에서 브랜드에 대한 감정을 추적하여 이미지를 관리하고, 부정적인 피드백이 발생했을 때 신속하게 대응할 수 있습니다. 금융 분야에서는 뉴스 기사와 소셜 미디어의 감성 분석을 통해 주가 예측, 시장 분석, 투자 전략 수립 등에 활용하며, 투자자들이 특정 기업이나 산업에 대해 어떻게 느끼는지 파악합니다. 정치 분야에서는 소셜 미디어와 뉴스 기사에서 정치적 사건에 대한 대중의 반응을 분석하고, 선거 기간 동안 후보자나 정책에 대한 대중의 감정을 모니터링합니다.

**1. 데이터 전처리**

**1.1 데이터 로드 및 결측값 처리**

와인 리뷰 데이터를 CSV 파일로부터 로드하고, 결측값을 포함한 행을 제거했습니다. reviews.rating과 reviews.text 칼럼에 결측값이 있는 행을 제거하여 정확한 분석을 위한 데이터 셋을 준비했습니다.

# 데이터 로드 및 전처리 함수 정의

file\_path = 'wine\_review.csv'

wine\_data = pd.read\_csv(file\_path)

# 리뷰 점수와 텍스트의 결측값 제거

wine\_data = wine\_data.dropna(subset=['reviews.rating', 'reviews.text'])

**1.2 리뷰 점수 변환**

리뷰 점수를 강한 부정, 약한 부정, 보통, 긍정, 강한 긍정의 다섯 가지 카테고리로 변환했습니다. 이를 통해 감정 분석이 용이해졌습니다.

# 리뷰 점수를 감정으로 매핑

rating\_mapping = {1: 'Strong Negative', 2: 'Weak Negative', 3: 'Neutral', 4: 'Positive', 5: 'Strong Positive'}

wine\_data['sentiment'] = wine\_data['reviews.rating'].map(rating\_mapping)

**1.3 텍스트 전처리**

리뷰 텍스트를 소문자로 변환하고, 특수문자를 제거했으며, 불용어를 제거한 후 표제어 추출을 수행하여 텍스트 데이터를 정제했습니다.

# 텍스트 전처리 함수 정의

def preprocess\_text(text):

# 소문자로 변환

text = text.lower()

# 특수문자 제거

text = re.sub(r'[^a-zA-Z\s]', '', text)

# 토큰화

words = text.split()

# 불용어 제거

stop\_words = set(stopwords.words('english'))

words = [word for word in words if word not in stop\_words]

# 표제어 추출

lemmatizer = WordNetLemmatizer()

words = [lemmatizer.lemmatize(word) for word in words]

return ' '.join(words)

**2. 모델 구현**

**2.1 데이터 분할**

전처리를 적용하고 데이터를 학습 세트와 테스트 세트를 80:20로 분할하여 모델의 성능을 평가할 준비를 했습니다.

# 모든 리뷰 텍스트에 전처리 적용

wine\_data['cleaned\_text'] = wine\_data['reviews.text'].apply(preprocess\_text)

# 특성 및 레이블 설정

X = wine\_data['cleaned\_text']

y = wine\_data['sentiment']

# 데이터 분할 (훈련/테스트 세트)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**2.2 텍스트 벡터화**

CountVectorizer를 사용하여 텍스트 데이터를 벡터화했습니다.

# 텍스트 데이터 벡터화

vectorizer = CountVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

**2.3 모델 학습 및 하이퍼파라미터 튜닝**

로지스틱 회귀 모델을 학습하고, GridSearchCV를 사용하여 최적의 하이퍼파라미터를 찾았습니다.

# 로지스틱 회귀 모델 학습 및 하이퍼파라미터 튜닝

model = LogisticRegression(max\_iter=1000)

param\_grid = {'C': [0.1, 1, 10, 100]}

grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train\_vec, y\_train)

# 최적 모델 선택

best\_model = grid\_search.best\_estimator\_

**3. 모델 평가**

**3.1 교차 검증**

교차 검증을 사용하여 모델의 견고성을 평가했습니다.

# 교차 검증을 통한 성능 평가

cv\_accuracy = cross\_val\_score(best\_model, vectorizer.transform(X), y, cv=5, scoring='accuracy')

cv\_precision = cross\_val\_score(best\_model, vectorizer.transform(X), y, cv=5, scoring='precision\_weighted')

cv\_recall = cross\_val\_score(best\_model, vectorizer.transform(X), y, cv=5, scoring='recall\_weighted')

cv\_f1 = cross\_val\_score(best\_model, vectorizer.transform(X), y, cv=5, scoring='f1\_weighted')

print(f"Cross-Validated Accuracy: {cv\_accuracy.mean()} (+/- {cv\_accuracy.std() \* 2})")

print(f"Cross-Validated Precision: {cv\_precision.mean()} (+/- {cv\_precision.std() \* 2})")

print(f"Cross-Validated Recall: {cv\_recall.mean()} (+/- {cv\_recall.std() \* 2})")

print(f"Cross-Validated F1 Score: {cv\_f1.mean()} (+/- {cv\_f1.std() \* 2})")

Accuracy: 0.8650306748466258

Precision: 0.8141061690524882

Recall: 0.8650306748466258

F1 Score: 0.8151763016430865

ROC AUC Score: 0.8449611349876361

**3.2 성능 지표**

모델의 성능을 평가하기 위해 정확도, 정밀도, 재현율, F1 점수 및 ROC-AUC 점수를 계산했습니다.

# 모델 평가

y\_pred = best\_model.predict(X\_test\_vec)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

roc\_auc = roc\_auc\_score(label\_binarize(y\_test, classes=best\_model.classes\_), best\_model.predict\_proba(X\_test\_vec), multi\_class='ovr')

# 결과 출력

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

print(f"ROC AUC Score: {roc\_auc}")

**4. 시각화 및 결과 분석**

**4.1 혼동 행렬**

혼동 행렬을 시각화하여 모델이 각 클래스를 어떻게 분류했는지 확인했습니다.

# 혼동 행렬 시각화

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=best\_model.classes\_, yticklabels=best\_model.classes\_)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

**4.2 ROC 곡선**

각 클래스에 대한 ROC 곡선을 시각화하여 모델의 분류 성능을 평가했습니다.

# ROC 곡선 시각화

y\_test\_binarized = label\_binarize(y\_test, classes=best\_model.classes\_)

plt.figure(figsize=(10, 7))

for i, sentiment in enumerate(best\_model.classes\_):

fpr, tpr, \_ = roc\_curve(y\_test\_binarized[:, i], best\_model.predict\_proba(X\_test\_vec)[:, i])

plt.plot(fpr, tpr, label=f"{sentiment} (AUC = {roc\_auc\_score(y\_test\_binarized[:, i], best\_model.predict\_proba(X\_test\_vec)[:, i]):.2f})")

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.show()

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

**5. 분석 및 결론**

모델은 전반적으로 높은 정확도와 재현율을 보였으며, F1 점수도 비교적 높았습니다. ROC 곡선 및 AUC 점수를 통해 모델이 각 클래스에서 어떻게 성능을 발휘하는지 확인할 수 있었습니다. 긍정적인 리뷰와 강한 긍정 리뷰에서는 높은 성능을 보였으나, 약한 부정 리뷰와 보통 리뷰에서는 상대적으로 낮은 성능을 보였습니다. 이는 강한 표현 같은 경우는 명백하게 구분할수 있으나, 약한 표현이나 중립과 같은 표현은 잘 알아차리기 힘들다는것을 알수있습니다.