기계학습 프로젝트

10조 | 202184001 강다현 / 202302145 김서진

목차

01 프로젝트 설명

프로젝트 주제: 아마존 리뷰& 감정 분석

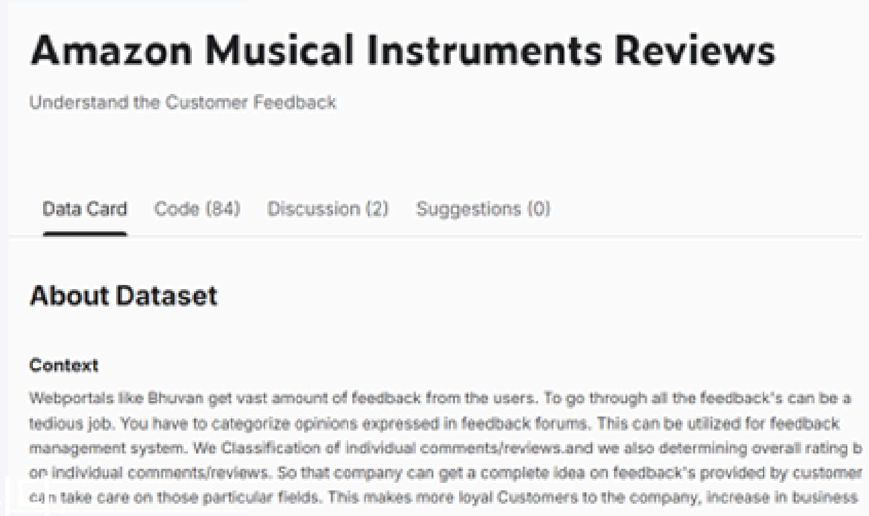
-아마존 리뷰에 있는 감정을 긍정 중립 부정으로 분석하고, word2vec을 이용해 단어 의미 벡터 생성하며 word2vec과 ml을 결합해 실험 및 성능 비교

수정 사항 | 리뷰 감정을 분노,혐오, 공포, 행복, 슬픔, 놀람 등으로 더 상세히 분석할 예정이었으나 해당 모델이 임베딩 벡터가 아닌 raw_Text만 받아 제외

02 사용한 데이터셋 소개

데이터셋

Kaggle - **Amazon Musical Instruments Reviews**



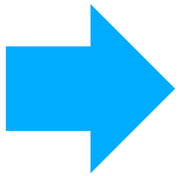
비율을 차지하고 있습니다.

• 구성: 리뷰 작성자, 제품 ID, 리뷰 본문(reviewText), 요약(summary), 평점(overall), 도움(helpful), 작성일(reviewTime)

03 구현 사항

1. stemming 제거: tfidf와 달리 word2vecd은 문맥을 파악해 벡터를 만드는 모델이므로 단어 형태를 통일해서 의미를 단순화시키면(stemming) 문맥적 차이를 학습하지 못함.

```
#Performing stemming on the review dataframe
ps = PorterStemmer()

#splitting and adding the stemmed words except stopwords
corpus = []
for i in range(0, len(review_features)):
    review = re.sub('[^a-zA-Z]', ' ', review_features['reviews'][i])
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stop_words]
    review = ' '.join(review)
    corpus.append(review)
```



```
corpus = []
for i in range(0, len(review_features)):
    review = re.sub('[^a-zA-Z]', ' ', review_features['reviews'][i])
    review = review.lower().split()        # 소문자 + 단순 split
    corpus.append(review)                  # 리스트 형태 그대로 저장
```

2. word2vec 임베딩 벡터 매개변수 설정(word2vec 임베딩 모델 학습시키는 코드)

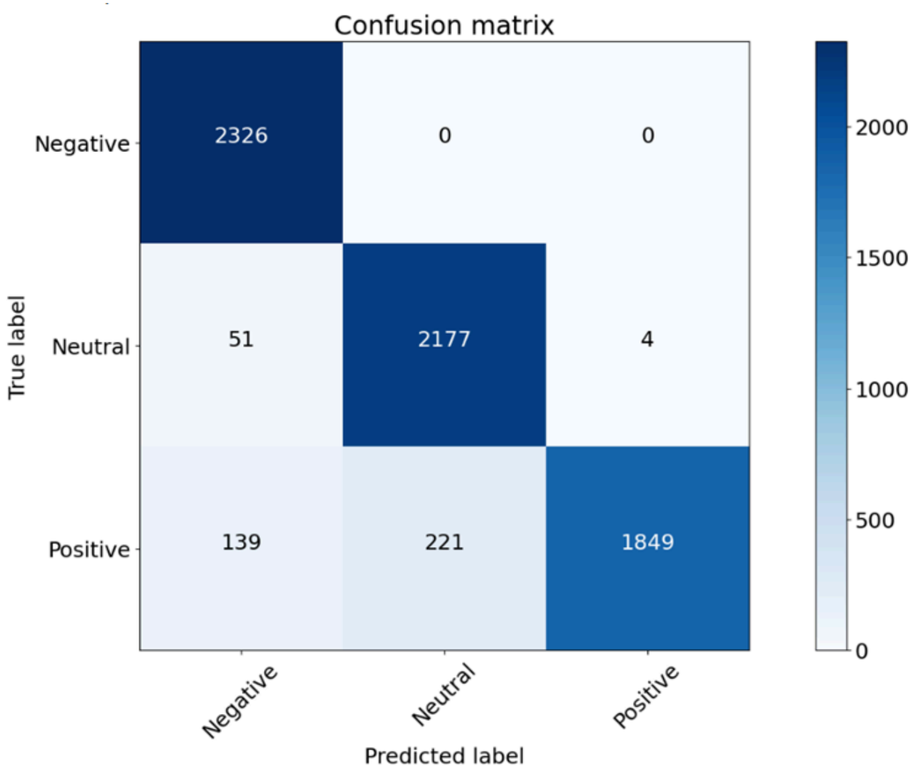
```
w2v_model = Word2Vec(
    sentences=corpus,    # 학습에 사용할 문장 데이터 (토큰화된 단어 리스트)
    vector_size=100,     # 단어 벡터의 차원 수 (각 단어를 100차원의 벡터로 표현)
    window=5,            # 중심 단어 기준으로 앞쪽 몇 단어까지 문맥으로 볼지
    min_count=2,         # 최소 등장 횟수. 2번 이상 등장한 단어만 학습에 포함
    workers=4,           # CPU 병렬 처리 스레드 수 (빠른 학습을 위해 4개 사용)
    sg=1                 # 학습 알고리즘 선택: 0=CBOW, 1=Skip-gram
)
```

```
def reviews_to_w2v_vectors(reviews, w2v_model):
    """
    여러 리뷰를 Word2Vec 임베딩 평균 벡터로 변환하는 함수

    Parameters
    ----------
    reviews : list of str
        리뷰 문장들의 리스트
    w2v_model : gensim.models.Word2Vec
        학습된 Word2Vec 모델

    Returns
    -------
    np.ndarray
        모든 리뷰의 벡터 (리뷰 수, 벡터 차원)
    """
    all_vectors = []

    for review in reviews:
        tokens = review.split()  # 리뷰를 단어 단위로 분리
        vecs = [w2v_model.wv[word] for word in tokens if word in w2v_model.wv]
        if len(vecs) == 0:
            # 리뷰에 단어가 없거나 OOV 단어만 있는 경우 0 벡터 반환
            all_vectors.append(np.zeros(w2v_model.vector_size))
        else:
            # 단어 벡터 평균 내기
            all_vectors.append(np.mean(vecs, axis=0))

    return np.array(all_vectors)
```

3. 각 리뷰를 → Word2Vec 단어 벡터들의 평균 벡터로 바꿔서 → 머신러닝 모델이 사용할 수 있는 숫자 벡터로 만드는 작업

04 모델 평가

TFIDF 임베딩 벡터를 통한 모델 학습




```
Logistic Regression Test Accuracy: 0.8807135224190521
Decision Tree Test Accuracy: 0.8149308817863116
KNN Test Accuracy: 0.8748661386236337
SVC Test Accuracy: 0.8796413976627168
Naive Bayes Test Accuracy: 0.802259082738763
```

```
print("Classification Report:\n",classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      2326
           1       0.91      0.98      0.94      2232
           2       1.00      0.84      0.91      2209

    accuracy                           0.94      6767
   macro avg       0.94      0.94      0.94      6767
weighted avg       0.94      0.94      0.94      6767
```

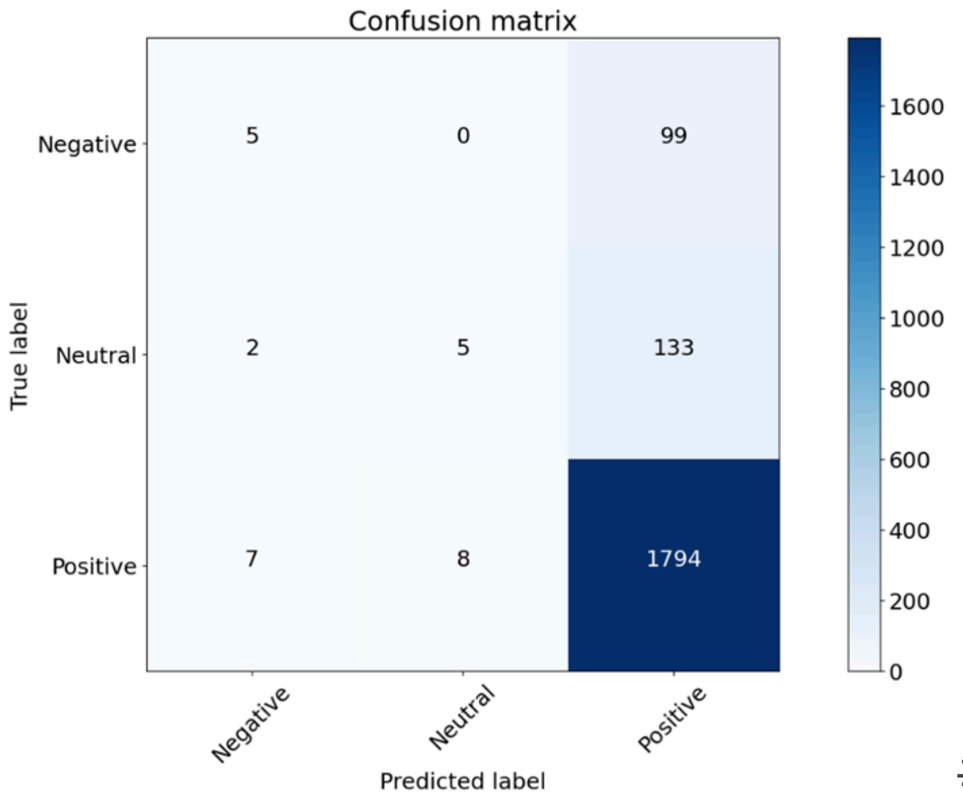
전체적으로 부정,중립,긍정을 분류하는 정확도가 높다. 부정 탐지 능력이 가장 탁월하며, 긍정을 가장 덜 정확하게 분류하고, 대체적으로 분류를 잘한다.

04 모델 평가

word2vec 임베딩 구현을 통한 모델 학습-cbow

```
[80]: cm = metrics.confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm, classes=['Negative','Neutral','Positive'])
```

```
Confusion matrix, without normalization
```




```
Logistic Regression Test Accuracy: 0.8787642046802606
Decision Tree Test Accuracy: 0.7794564307555647
KNN Test Accuracy: 0.8634636737901227
SVC Test Accuracy: 0.8792515341149585
Naive Bayes Test Accuracy: 0.77848253111411
```

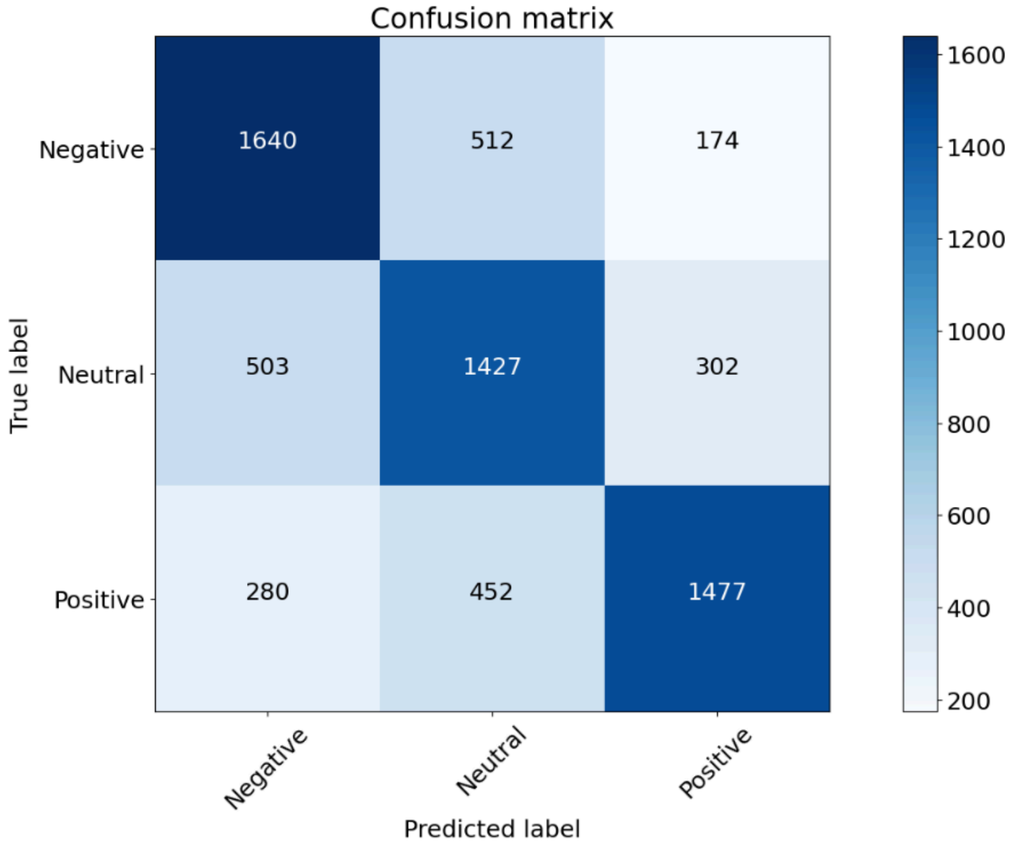
```
[81]: print("Classification Report:\n",classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.36      0.05      0.08       104
           1       0.38      0.04      0.07       140
           2       0.89      0.99      0.94      1809

    accuracy                           0.88      2053
   macro avg       0.54      0.36      0.36      2053
weighted avg       0.82      0.88      0.83      2053
```

극심한 데이터 불균형-Accuracy 0.88은 좋은 모델이 아니라
positive 데이터가 불균형하게 많아 생긴 허상
모델이 Positive만 잘 맞추고,
Negative / Neutral은 거의 “분류 불능” 상태.

04 모델 평가

word2vec 임베딩 구현을 통한 모델 학습-skip-gram




```
Logistic Regression Test Accuracy: 0.88
Decision Tree Test Accuracy: 0.79
KNN Test Accuracy: 0.86
SVC Test Accuracy: 0.88
Naive Bayes Test Accuracy: 0.87
```

```
print("Classification Report:\n",classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.71      0.69      2326
           1       0.60      0.64      0.62      2232
           2       0.76      0.67      0.71      2209

    accuracy                           0.67      6767
   macro avg       0.68      0.67      0.67      6767
weighted avg       0.68      0.67      0.67      6767
```

tfidf에 비하면 정확도가 떨어지지만, wordvec 임베딩으로 구현시 cbow를 skip-gram으로 바꾸기만 한 것으로도 방금 전 cbow에 비해 모델 성능 과대평가가 사라진 것을 알 수 있다.

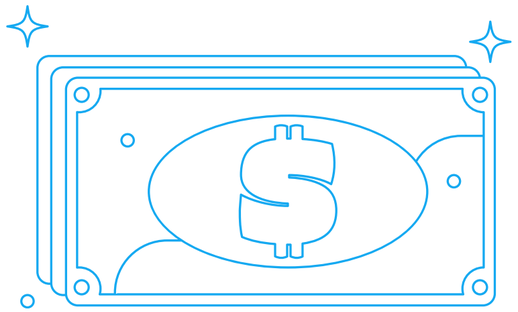
결론

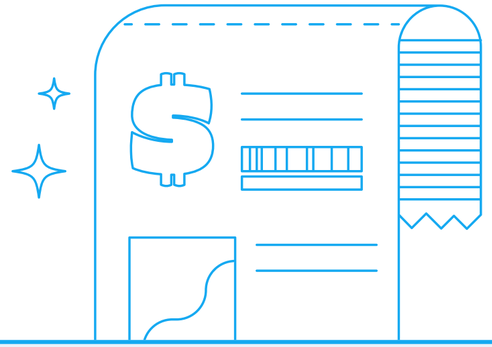
1. word2vec-cbow로 학습시킬 때는 positive 데이터가 많기 때문에 positive로 분류해 실제로는 모델 성능 과대평가가 일어남

2. CBOW와 Skip-gram은 학습 방식이 정반대인데, 감성 분석(Sentiment Analysis)과 같은 의미 파악 문제에서는 Skip-gram이 더 강력한 성능을 발휘->
   CBOW: 주변 단어 4개를 묶어서 -> 중심 단어 1개를 맞추는 문제로 만듬. (학습 기회 1회)
   Skip-gram: 중심 단어 1개로 -> 주변 단어 4개를 각각 맞추는 문제가 됨 (학습 기회 4회)
   결국 같은 문장을 가지고도 Skip-gram이 훨씬 더 많은 학습 샘플(Word Pair)을 만들어내기 때문에, 데이터를 더 샅샅이 훑어보는 효과가 있음

3. 최종적으로는 tfidf가 word2vec보다 해당 데이터셋에서 감정 분석에서는 더 좋은 성능을 보임.

05 데모 시연





STEP 01
TFIDF

STEP 02
cbow

STEP 03
skip-gram

감사합니다
THANK YOU