9 조 Mid-Report

1. 문제 정의

오늘날 대다수의 소비자들은 구매 의사 결정 상황에서 리뷰를 읽는다. 온라인 리뷰란 제품이나 서비스를 구매한 후, 이를 경험한 고객이 해당 제품(서비스)에 대해 평가한 것을 의미한다. 본 프로젝트에서는 이러한 제품의 텍스트 리뷰 데이터들로부터 1) 제품의 잠재적인 제품 속성을 추출하고, 이를 바탕으로 전체 텍스트 리뷰를 대표하는 2) 토픽(핵심어)을 생성해 3) 각 리뷰들을 분류해볼 것이다. 이를 통해, 리뷰를 읽는 사용자가 효율적으로 자신의 구매 조건에 맞는 리뷰들을 선별해 읽을 수 있도록 하는 것이 목표이다.

2. 시도해본 방법론 및 중간 결과 (파이프라인)

2.1. 데이터셋 선정 및 전처리

처음에는 영화, 호텔, 특정 상품 리뷰와 같이 서로 다른 성격의 제품 리뷰들이 혼합된 텍스트 데이터셋을 활용하고자 하였다. 그러나 문제 정의를 구체화하는 과정에서, 제품별 클러스터를 구분하는 단계는 프로젝트의 목적 상 불필요하다고 판단하였다. 그 이유는 실생활에서 리뷰 데이터는 일반적으로 하나의 제품을 단위로 하여 그에 대해서 제공되는 경우가 많기 때문이었다. 이에 따라 본 프로젝트에서는 아마존의 Beauty (화장품 및 미용용품) 카테고리에 해당하는 단일 텍스트 리뷰만을 사용하여, 해당 카테고리의 핵심 속성을 추출하기로 하였다.

Amazon Review Data [1]는 2014 년에 등록된 아마존*(Amazon)*의 온라인 리뷰 데이터셋이다. 해당 데이터셋은 텍스트 리뷰 데이터뿐만 아니라 별점과 같은 리뷰들을 포함하고, 또 카테고리 정보나 설명과 같은 메타데이터들을 담고 있다. 본 프로젝트에서는 리뷰 기반 속성 추출을 위하여 오직 텍스트 리뷰 데이터만을 사용한다. 그리고 해당 데이터셋에서 제공되는 여러 상품 카테고리들 중, 속성의 종류가 비교적 다양하게 추출될 것이라 기대되는 Beauty 카테고리의 상품 리뷰 데이터만을 수집하였다.

해당 데이터셋을 처리하기 위하여 다음과 같은 데이터 전처리 작업을 수행하였다. *nltk* 라이브러리의 *TreebankWordTokenizer* 를 사용하여 문장으로 이루어진 리뷰를 토큰화 하였다. 이후 (파이프라인의 최종 단계인) 토픽 모델링 작업을 수행하기 위해서는 표제어를 추출해야 했기 때문에 각 토큰들을 모두 기본형으로 바꿔주었다. 마지막으로는, 데이터 처리에 방해가 될 수 있는 여러 불용어들을 제거한 후, 37 만여개의 뷰티 데이터에서 오직 3 만 개의 데이터만을 랜덤 샘플링해 최종 데이터셋을 구성하였다.

2.2. 문서 임베딩

리뷰 데이터를 본격적으로 분석하기에 앞서 텍스트 데이터(문서)를 임베딩하는 작업을 수행하였다. 텍스트 데이터는 대부분 고차원이기 때문에, 이러한 데이터를 그대로 사용하게 되면 높은 계산 비용이 발생할 뿐만 아니라 유사성 측정 및 결과 해석 등에서 어려움을 겪을 수 있다.

클러스터링 성능 또한 저하될 가능성이 높다. 이러한 문제들을 피하기 위해 본 프로젝트에서는 텍스트 데이터를 수치적으로 임베딩 하였다.

2.2.1. 시행착오 및 결과

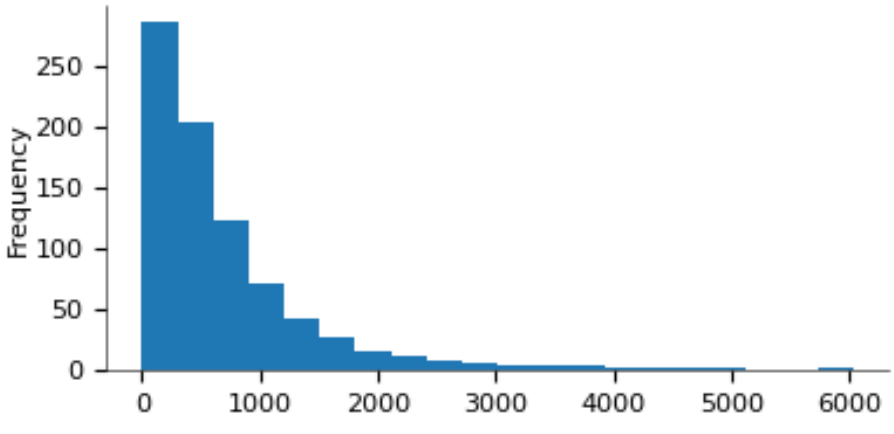
문서를 임베딩하는 방법에는 여러가지가 존재하나, 본 프로젝트에서는 텍스트 마이닝에서 일반적으로 많이 활용되는 TF-IDF 방법을 사용하여 문서 임베딩 작업을 수행하였다. 이때 TF 는 'Term Frequency'로 단어 빈도, 다시 말해 문서 내에서 각 단어가 얼마나 자주 등장하는지를 측정하는 지표를 의미하며, IDF 는 'Inverse Document Frequency'로 역문서 빈도, 다시 말해 특정 단어가 여러 문서에서 공통적으로로 등장하는 정도를 나타내는 지표를 의미한다. 특히 이것은 특정 단어가 얼마나 일반적인지를 나타낸다. TF-IDF 는 이 두 가지 지표를 결합하여 단어의 중요성을 점수로 계산한 후, 이 값을 바탕으로 텍스트 문서를 임베딩하는 기법이다.

해당 임베딩 기법에서는 'max_features' 매개변수가 중요한 하이퍼 파라미터로서 최종 임베딩 벡터의 단어 수를 설정하는 데 사용된다. 다시 말해 이는 TF-IDF 변환 과정 중에 선택할 최대 단어 수(등장 횟수 기준 상위 몇개까지의 단어를 남길 것인지)를 결정한다. 이 값을 통해 단어 수를 제한하면, 이에 따라 TF-IDF 벡터의 차원이 결정된다. 따라서 너무 작은 값을 설정하면 정보 손실이 발생할 수 있으며, 너무 큰 값을 설정하면 계산 및 메모리 문제가 발생할 수 있다.

가장 첫 번째 시도로는 max_features 의 값을 임의로 20000 으로 지정하여 임베딩한 후 차원축소 과정을 수행해 보았다. 그런데 해당 값으로 생성된 임베딩 벡터는 차원 축소하는 데 시간이 너무 오래 소요된다는 문제가 발생했다. 이에 팀에서는 원 데이터셋에 대한 EDA 를 통해 데이터셋 내 단어별 등장 횟수를 파악하고, 이에 기반하여 적절한 max_features 값을 찾는 방식을 고안하였다.

```
0 ('use', 29760)
1 ('hair', 22880)
2 ('product', 20294)
3 ('skin', 15840)
4 ('like', 15018)
5 ('get', 12289)
6 ('good', 10028)
7 ('one', 9468)
8 ('make', 9461)
9 ('work', 9158)
10 ('great', 9106)
11 ('love', 8970)
12 ('look', 8885)
13 ('really', 8645)
14 ('would', 7930)
15 ('try', 7680)
16 ('smell', 7561)
17 ('dry', 7470)
18 ('go', 7244)
19 ('time', 7074)
20 ('color', 6972)
21 ('buy', 6759)
22 ('well', 6619)
23 ('face', 6047)
24 ('feel', 5957)
25 ('day', 5932)
26 ('also', 5707)
27 ('much', 5592)
28 ('little', 4995)
29 ('long', 4987)
30 ('think', 4830)
31 ('even', 4791)
32 ('find', 4701)
33 ('it', 4668)
34 ('leave', 4565)
35 ('give', 4390)
36 ('brush', 4255)
37 ('say', 4249)
38 ('recommend', 4229)
39 ('apply', 3917)
40 ('price', 3909)
```

[등장 순위 상위 TOP40 단어]



[단어별 등장 빈도]

등장빈도 순으로 데이터셋의 모든 토큰화된 단어를 정렬해본 결과, 상위 20 개에 해당하는 단어와 그 등장빈도는 다음 그림과 같았다. 그리고 여기에서 기존에 값으로 지정하였던, (상위) 2 만째에 해당하는 단어의 등장 빈도를 확인해보았다. 그 결과, 상위 2 만번째의 단어는 등장빈도가 매우 작았기에 이렇게 적은 등장빈도의 단어까지 벡터화에 고려할 필요는 없다고 보았다. 이에 본팀에서는 max_features 값을 20000 에서 더 줄이기로 하였다. 구체적인 값을 설정하기 위해서는 데이터셋 내 단어별 등장빈도가 달라지는 구간을 파악할 필요가 있었고, 이에 따라 아래와 같이 등장빈도를 시각화하여 전체적인 데이터 경향성을 파악하였다. 그리고 이러한 결과를 바탕으로, 등장 빈도가 대략 9-10 회 이상인 단어만을 벡터화에에 고려하는 것이 유의미하다고 판단하여 max_features 의 값을 6000 으로 설정하였다.

2.3. 차원축소

클러스터링을 위해서는 차원 축소가 필수적이다. 차원 축소는 고차원의 데이터를 사람이 인식할 수 있는 저차원으로 데이터를 시각화할 수 있어, 클러스터 간의 관계를 이해하기 쉽다. 뿐만 아니라, 차원의 저주(curse of dimensionality)를 피할 수 있다. 고차원의 데이터일수록, 데이터 간의 거리가 멀고, 밀도도 감소하기 때문에, 효율적인 클러스터링을 하기 위해서 차원 축소를 해야만 한다. 차원 축소 기술은 다양하게 존재하므로, 본 팀에서는 각 장단점을 고려해 적절한 기법을 선택하고자 하였다.

2.3.1. 차원 축소 시행착오 및 결과

주성분 분석(Principal Component Analysis, PCA)은 대표적인 차원 축소 기술 중 하나이다. 빠르고 해석이 쉬워서 많이 사용되었기에 가장 첫 번째 방법으로 사용하였다. PCA 는 주로 선형성을 띠는 경우 사용하는데, PCA 는 데이터의 분산을 최대화하는 낮은 차원 하위 공간을 찾아 선형 투영을 수행한다. 이것은 데이터가 가능한 한 퍼져 있게(분산을 최대화하는 방향으로) 만드는 것을 의미하되, 데이터 포인트 간의 관계는 변화시키지 않도록 한다. 그 결과, 데이터 포인트가 어떠한 군집에 존재하는지 식별할 수 있게 해준다.

그렇기에, 가장 먼저 시도해본 차원 축소 기법이 바로 PCA 이었다. 그러나, 앞선 문서 임베딩에서 TF-IDF 를 채택하였는데, TF-IDF 가 sparse matrix 로 구성되기 때문에 covariance matrix 가 정밀도 손실을 일으킬 수 있다. 또한, 선형으로 투영하기 때문에 클러스터들이 뭉개진다는 단점의 영향이 컸다. 따라서, 차원 축소 기법으로 PCA 를 도입할 수 없다는 결론에 다다랐다.

따라서, sparse matrix 를 사용하기 위해서 PCA 같은 또 다른 차원 축소 방법인 SVD 를 사용해보았다. 특이값 분해 (Singular Value Decomposition, SVD)은 선형 대수학에서 사용되는 주요 기법 중 하나이다. SVD 는 원본 행렬을 3 개의 부분으로 분해하는 과정을 거친다. 분해되는 세 행렬은 각각 U, V, T 인데, 차례대로 원본 데이터의 행간 상관관계, 특이값 대각 행렬, 열간 상관관계를 나타내는 행렬들이다. SVD 를 이용해 상위 feature 들을 선택해주고, 대응되는 특이값과 열을 선택하여 데이터 차원을 줄여주는 과정이다. 이렇게 해줄 경우, 낮은 랭크의 근사 행렬로 분해해주어 특성을 추출할 수 있다. 중요한 구조 및 특성을 유지하며 저차원으로 투영할 수 있으며, 효율성도 뛰어나다.

SVD 를 구현하기 위하여 scikit-learn 의 TruncatedSVD 를 사용해주었다.

```
dim_reduction_model = TruncatedSVD(n_components=n_components,
  random_state=config.random_state)
dim_reduced = dim_reduction_model.fit_transform(tfidf_vec)
```

이때 SVD 를 기준으로 PoV 가 0.9 보다 커지는 경우의 최적의 n_components 를 찾고자 하였고, 그 결과 3500 차원일 경우에 해당 조건을 만족시킴을 확인하였다.

또 다르게 시도해 본 방법으로는 t-SNE 가 있다. t-SNE(t-distributed Stochastic Neighbor Embedding)는 데이터 포인트 간의 유사성을 보존함과 동시에 고차원 데이터를 저차원으로 투영하는 대표적인 비선형 차원 축소 기법이다. 그렇기 때문에 복잡한 구조나 비선형 관계를 잘 보존하고, 특히나 데이터셋이 고차원일 경우, 시각화하는데에서 성능이 두드러진다. 그렇기 때문에 PCA 와 같은 선형 차원 축소가 유의미한 결과를 내기 어려운 복잡한 구조의 데이터에서 유리하다.

t-SNE 는 scikit-learn 의 manifold.TSNE 를 통해 다음과 같이 구현할 수 있다.

```
tnse = TSNE(n_components=3, random_state=config.random_state)
tnse_reduced = tnse.fit_transform(dim_reduced)
```

그러나 이 방법 같은 경우는, 데이터의 개수를 n 이라 하였을 때 시간 복잡도가 O(n^2) 이기 때문에 동작에 상당히 오랜 시간이 소요된다. 뿐만 아니라, n_components 로 4 보다 작은 값만을 지정할 수 있어 너무 높은 차원을 바로 축소시키는 것이 어렵다고 볼 수 있다. 더불어, training 과 prediction 을 동시에 수행하기에 매 시행마다 서로 다른 시각화가 나오기에 deterministic 하지 못하다고도 볼 수 있다.

따라서 높은 시간 복잡 및 sparsity 로 인한 정밀도 손상 문제를 극복하기 위하여, 본 팀에서는 결론적으로 TruncatedSVD 를 차원 축소 기법으로 사용하기로 결정하였다.

2.4. 클러스터링

2.4.1. 시행착오

클러스터링에 대한 첫 번째 시도로서, 가장 직관적이고 간단한 K-Means Clustering 알고리즘을 사용해보았다. 이는 임베딩 된 문서들 간의 거리 값을 이용하여 클러스터를 생성하는 방법론인데, 우선은 클러스터의 개수를 임의로 10 개로 지정하여 알고리즘의 성능을 테스트해보려 하였다.

2.4.1.1 K-Means

본 팀에서 실험 전 우려했던 부분은 K-Means 알고리즘이 non-deterministic 한 알고리즘이라는 점이었다. 해당 알고리즘은 그 initial centroid 가 어디에 설정되는지에 따라 클러스터링 결과가 변하게 된다. 그런데 클러스터링의 결과가 매번 달라지게 되면, 그에 기반하여 생성되는 각 문서 별 토픽이 달라질 수 있게 되는데, 이는 본 프로젝트에서 정의한 문제의 해결에 적합하지 않다. 따라서 본 팀에서는 K-means 클러스터링 알고리즘을 사용하는 과정에서 고정된 random seed 값을 설정하고자 하였다. 그런데 이렇게 임의로 설정한 random seed 값을 사용하게 되면, 전체 파이프라인(문서 분류 및 토픽 추출)의 최적 하이퍼파라미터를 찾는 데 있어 영향을 줄 수 있다는 점이 우려되었다. 데이터의 구조가 아닌 임의로 설정한 random seed 값이 영향을 미치는 것은 적절하지 않다고 보았기 때문이다. 따라서 본 팀에서는 그 영향이 얼마나 될지 알아보기 위해, random seed 를 고정하지 않은 상태에서 클러스터링을 5 회 시행해보았다.

```
 clustering_model = KMeans(n_clusters=10)
largest_cluster_counts = []

for i in range(0, 5):
clustered = clustering_model.fit_predict(dim_reduced)
largest_cluster_counts.append(pd.Series(clustered).value_counts().iloc[0])

print(largest_cluster_counts)
del largest_cluster_counts

output: [16226, 16112, 20985, 16177, 16396]
```

각 iteration 에서 만들어진 가장 큰 클러스터의 크기를 모아 출력해보았고, 그 결과 각 클러스터 크기의 편차가 적지 않은 것이 확인되었다. 이에 따라 본 팀에서는 random seed 값이 전체 파이프라인의 성능 분석 및 최적화 과정에 상당한 영향을 미칠 것이라고 판단하여 이에 K-Means 대신 deterministic 한 클러스터링 알고리즘을 사용하기로 결정하였다.

2.4.1.2 Agglomerative

Deterministic 한 클러스터링 알고리즘 중 가장 먼저 시도한 것은 Hierarchical Clustering 알고리즘의 한 종류인 Agglomerative 알고리즘이었다. Agglomerative Clustering 알고리즘은 각각의 데이터 포인터들을 하나의 클러스터로 지정하고, 지정된 개수의 클러스터가 남을 때까지

가장 비슷한 두 클러스터를 합쳐 나가는 방식의 상향식 알고리즘이다. 이때 두 클러스터를 합쳐 나가는 방식에는 모든 클러스터 내의 분산을 가장 작게 증가시키는 두 클러스터를 합치는 방식(Ward), 클러스터 포인트 사이의 평균 거리가 가장 짧은 두 클러스터를 합치는 방식(Average), 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 합치는 방식(Complete) 등이 있다.

그러나 이 알고리즘의 경우에는 지나친 메모리 사용량을 인해 주어진 개발 환경에서 동작하지 않았다. 해당 알고리즘이 공간복잡도 O(n^2) 의 고비용 알고리즘이었기 때문이다. 또한 총 몇 개의 클러스터로 나눌 것인지, 해당 알고리즘의 하이퍼파라미터 K 를 결정하는 과정에서 Dendrogram 을 그리는 작업조차 주어진 개발 환경에서 지나친 공간 비용으로 인해 작업이 정상적으로 완료되지 않았다. 이에 본 팀에서는 알고리즘 수행은 물론, K 값조차 찾을 수 없었다. 이를 통해 팀에서는 Agglomerative Clustering 알고리즘이 본 프로젝트의 개발 환경 및 데이터셋의 크기 상 부적합하다고 판단하고, 이에 본 알고리즘보다 비교적 작은 공간 복잡도의 deterministic 알고리즘을 다시 찾기 시작했다.

2.4.1.3 DBSCAN

Deterministic 한 새로운 알고리즘을 찾는 과정 중, 일반적으로 모든 클러스터링 알고리즘에서 요구되는 하이퍼파라미터 K(몇 개의 클러스터로 나눌 것인가)에 대한 논의가 발생하였다. 구체적으로는 최적의 K 값을 찾기 위한 메트릭 지표를 분석하던 중, 임베딩 및 차원축소 된 데이터의 구조를 정확히 알지 못하는데, 그 상태에서 K 값을 현재 주어진 데이터에 기반하여 임의로 지정하는 것이 맞는가에 대한 의문점이 생겼다. 이에 데이터의 밀도에 따라 알고리즘에서 자체적으로 K 값을 결정해 클러스터링을 진행하는 DBSCAN 알고리즘을 사용해보게 되었다.

그런데 DBSCAN 알고리즘을 동작시키면서 앞선 Agglomerative 와 동일한 문제에 봉착했다. 높은 메모리 사용량으로 인해 주어진 개발 환경에서 동작이 되지 않았다. DBSCAN 알고리즘의 공간 복잡도 또한 O(n^2)로써 높은 편이기 때문이었다.[3] 따라서 같은 밀도 기반 클러스터링 알고리즘 중 이러한 문제를 개선한 HDBSCAN 알고리즘을 사용해보기로 결정하였다.

2.4.2. 결과

최종적으로는 HDBSCAN 클러스터링을 시도하였으나, 축소된 차원이 HDBSCAN 에 적합한 차원보다 훨씬 크다는 문제가 있어, 현재 파이프라인 상에서 이를 활용하기엔 시간이 너무 오래 걸렸다.[4] 하지만 팀에서는 이는 알고리즘의 문제라기보다는 데이터 임베딩의 문제라 판단하였다. 그 이유는 TruncatedSVD 에서 정보량 손실을 줄이기 위해서 PoV 가 90% 이상인 차원 축소 target 을 사용하려 하였기에 전처리된 데이터의 차원이 너무 커졌고, 이 때문에 HDBSCAN 이 의도한 대로 동작하지 못했기 때문이다.

오히려 HDBSCAN 은 K 값을 고를 때 임의적인 판단이 필요하지 않고, 차원 값만 낮춰준다면 메모리 문제, 시간 문제 모두 해결할 수 있기 때문에 클러스터링 알고리즘은 HDBSCAN 으로 고정하고, 문서 임베딩 방법을 바꿔보기로 하였다.

2.5. 토픽 모델링

토픽 모델링을 위해 사용되는 모델에는 여러 가지가 있는데, 가장 대표적인 것 중 하나는 LDA(Latent Dirichlet Allocation)이다. LDA 는 Bag-Of-Words 형태의 문서 표현으로부터 통계적인 추론을 거쳐 문서를 토픽 분포와 토픽 내 단어 분포의 조합으로 표현하는 모델이다. 구현이 쉽고 가장 기본적인 토픽 모델링 모델로 많이 사용되기에, 설계한 파이프라인에 쉽게 적용할 수 있는 모델이라 생각하여 선택하였다.

앞선 단계에서 클러스터링이 이루어지면, 각 클러스터 단위로 LDA 를 적용하여 비슷한 의미 단위로 묶인 문서 집합 내의 잠재적인 토픽을 추출한다. 이때 클러스터링을 적용하지 않고 전체 문서에 대해 LDA 를 수행한 것을 Baseline 모델로 하여 결과를 비교하며 클러스터링 방법론 자체에 대한 효과성 또한 검증해보려고 하였다.

추출된 토픽 분포를 바탕으로 문서별로 키워드를 추출하는데, 추출하는 방법에는 [5]에서 제안된 Term saliency 지표를 활용하기로 하였다. Term saliency 란 특정 단어 w 가 토픽 T 를 생성하는 데 얼마나 기여했는지(distinctiveness)와 그 단어의 등장 빈도(P(w))를 복합적으로 고려하여 생성한 지표이다. 많은 토픽에 반복해서 등장한 단어보다는 특정 토픽을 대표하는 단어들일수록 term saliency 가 높게 산정되므로, 여러 토픽에 걸친 단어들을 고루 추출해낼 수 있다는 장점이 있다. Term Saliency 기준 상위 30 개의 단어를 키워드로 선정한 뒤, 문서 내에 해당 키워드가 있으면 그 문서의 키워드로 설정하는 방식으로 문서별 키워드를 설정하였다.

$$distinctiveness(w) = \sum_{T} P(T|w) \log \frac{P(T|w)}{P(T)}$$

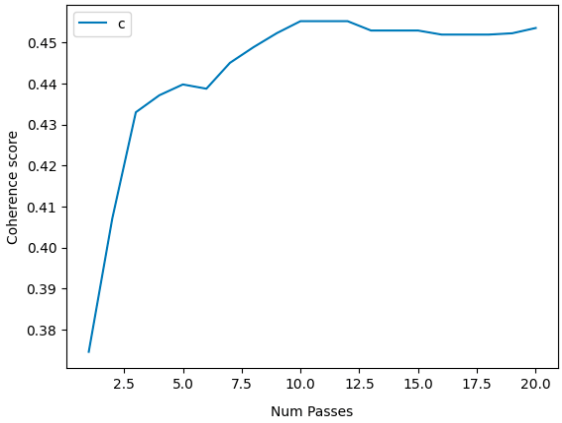
$$saliency(w) = P(w) \times distinctiveness(w)$$

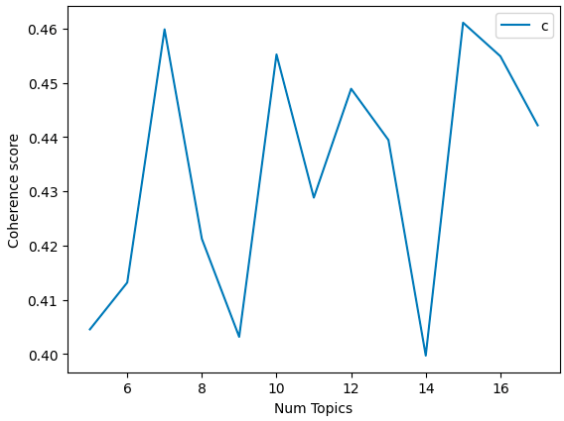
LDA 모델 자체의 하이퍼파라미터를 튜닝할 때에는 Topic coherence score 를 활용하였다. Topic coherence score 란 토픽이 얼마나 의미 있고 해석 가능한지를 나타내는 지표로, 각 토픽을 구성하는 단어들 간의 의미적 일관성을 측정한 지표이다.

2.5.1. 결과

앞선 단계에서 과도하게 큰 차원으로 인한 시간/공간 복잡도 문제로 클러스터링을 온전하게 수행할 수 없게 되었다. 따라서 중간 보고서 작성까지는 기존 Baseline 모델이었던 전체 문서를 대상으로 한 LDA 모델의 결과만 확인해볼 수 있었다.

먼저 Baseline 모델의 파라미터 튜닝을 수행하였다. passes 와 num_topics 파라미터를 대상으로 하였는데, passes 는 모델이 전체 문서를 몇 번 순회할 것인지를 결정하는 파라미터이고, num_topics 는 LDA 가 추출할 토픽의 개수를 결정하는 파라미터로 모두 성능에 결정적인 영향을 미치는 파라미터이다. 먼저 num_topics 를 10 으로 고정시킨 뒤 최적의 passes 를 찾았고, 이후 passes 를 구한 값으로 고정시킨 뒤 최적의 num_topcis 를 탐색하였다.





Topic Coherence
0.4598

최종적으로 선택한 모델의 Topic coherence 는 위와 같았다. 중간 보고 이후 다른 방법론을 적용했을 때의 topic coherence 점수를 위 baseline 모델의 지표와 비교하며 성능을 평가할 계획이다.

love	face	nail	coat	price	light
buy	scent	color	work	great	skin
hair	soft	conditioner	use	foundation	smell
look	brush	oil	acne	34	dry
cream	polish	like	lotion	shampoo	product

[추출된 토픽 키워드 목록]

리뷰	토픽 키워드
This leave-in is just what I was **looking** for to **use** on my naturally curly-coily **hair**. It is **great** for detangling and moisturizes without leaving build-up. Great for wash and go days!	great, look, hair, use

[리뷰와 추출된 키워드 예시]

토픽 모델링 수행 결과 전반적으로 price, color, skin, hair 등 뷰티 분야 리뷰에 관한 키워드들이 잘 추출된 모습을 보였다. 그러나 EDA 결과 파악되었던 curly, spray, shower 등 중심 토픽보다는 덜 등장하지만, '헤어 관련 리뷰'라는 세부 분야를 대표할 것으로 추정되는 단어들은 등장하지 않는 모습을 보였다. 클러스터링 없이 모든 리뷰를 대상으로 토픽 모델링을 수행하다보니 세부 토픽을 놓치게 되는 것으로 보였고, 클러스터링을 적용한 후 토픽 모델링을 수행할 경우 이러한 한계점을 보완할 수 있지 않을까 하는 가설을 세우게 되었다.

3. 개선해야 할 부분 및 개선 계획

3.1. 문서 임베딩

파이프라인 별 알고리즘을 평가하는 과정에서 두드러진 문제점은 임베딩한 데이터의 차원이 너무 크다는 것이었다. 차원 축소를 이용하여 raw 데이터보다 차원을 많이 줄였음에도, TF-IDF 가 가지는 sparse matrix 의 특징 때문에 기존 데이터를 잘 표현하기 위해서는 차원이 너무 많이 축소되지 않게끔 해야 했다. 이 때문에 HDBSCAN 알고리즘이 효율적인 시간 안에 실행되지 못했다. 따라서 우리는 문서 임베딩 알고리즘을 dense matrix 를 만들어낼 수 있는 것으로 바꾸려고 한다. 대표적으로는 Sentence BERT 가 있고, 딥러닝 기반 임베딩 모델들을 주로 사용하는 것으로 보여 더 조사해본 뒤 적합한 것을 찾아 실험해볼 계획이다.

3.2. 차원 축소

차원 축소 기법 중 하나인 t-SNE 의 다양한 단점 중에 치명적이었던 속도의 한계를 해결하기 위하여, 가장 좋은 성능을 내는 알고리즘으로 알려진 UMAP (Uniform Manifold Approximation and Projection)도 시도해보았다. UMAP 은 t-SNE 와 달리 임베딩 차원의 크기 제한이 별도로 없어 general 하게 적용할 수 있었고, 데이터의 전반적인 구조도 잘 보존할 수 있었다.

```
from umap import umap_ as UMAP
reducer = UMAP.UMAP(n_components=n_components,random_state=42)
embedding = reducer.fit_transform(tfidf_vec)
```

위와 같이 작성하여 UMAP 으로 차원 축소를 수행해보았다. 결론적으로 UMAP 은 데이터가 sparse 한 경우에도 적용 가능하고, 비선형의 복잡한 데이터 구조도 보존할 수 있었으며, 복잡도 측면에서도 우수한 성능을 보여주는 최적의 알고리즘이었다.
그러나 앞서 언급한 것처럼, 우선 문서 임베딩 알고리즘을 수정하고난 뒤 차원 축소 알고리즘을 고르기 위해 기존의 SVD 와 UMAP 을 비교 분석할 계획이다. Dense 한 임베딩에 포함된 선형/비선형 구조를 분석해 데이터에 더 적합한 차원 축소 기법을 확정할 수 있으리라 기대한다.

3.3. 클러스터링

앞서 클러스터링 알고리즘 실험에서 확인하였듯, 임베딩 차원을 줄이면 hdbscan 알고리즘을 적용할 수 있을 것으로 생각된다. HDBSCAN 알고리즘이 시간이 너무 오래 걸렸던 이유는 임베딩 된 문서 데이터가 차원이 너무 컸기 때문이다. 따라서 HDBSCAN 에 적합하게끔 문서 임베딩 결과 차원을 충분히 낮춘 뒤 클러스터링을 수행한다면 효율적인 시간 내에 문서를 클러스터링 할 수 있을 것이다. 이를 위해 수정된 문서 임베딩과 차원 축소 기법이 적용된 데이터에 대해 HDBSCAN 클러스터링을 수행하고, 성능 분석 메트릭을 조사 및 활용하여 HDBSCAN 의 두 hyperparameter 인 min_samples 와 min_cluster_size 를 tuning 해볼 예정이다.

3.4. 토픽 모델링

토픽 모델링의 경우 클러스터링이 완성되지 못하여 아직 클러스터 별 토픽 모델링을 수행해보지 못했으나, 문서 전체에 대해 토픽 모델링하는 파이프라인을 완성해두었기 때문에 클러스터링이 완성된다면 이를 그대로 적용할 수 있을 것이라 생각한다. 이후 문서 전체에 대한 토픽 모델링과의 성능 차이를 비교해보고, 어떻게 하면 클러스터 별 토픽 모델링 성능을 높일 수 있을지 방법을 조사해보고 성능을 개선할 수 있는 방안들을 실험을 통해 찾아볼 것이다.

4. Reference

[1] Justifying recommendations using distantly-labeled reviews and fined-grained aspects
Jianmo Ni, Jiacheng Li, Julian McAuley
Empirical Methods in Natural Language Processing (EMNLP), 2019
pdf

[2] https://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html

[3] https://github.com/scikit-learn/scikit-learn/issues/22531#issuecomment-1044424421

[4] https://hdbscan.readthedocs.io/en/latest/faq.html#q-i-am-not-getting-the-claimed-performance-why-not