

Suggestify - 초개인화 음악 추천 서비스 Au-Dionysos WrapUp Report

1. 프로젝트 개요

사용자의 취향을 담은 태그와 사용자의 감정과 상황을 담은 텍스트를 활용하여
취향 + 감정 + 상황을 모두 반영하는 개인 맞춤형 음악 추천 서비스

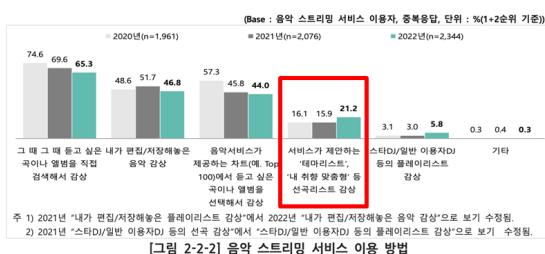
1.1 배경

1.1.1 초개인화 시대



지금은 이른바 부캐의 시대로, 한 사람의 내면 안에 시간, 장소 상황에 따른 다양한 자아가 존재한다.

1.1.2 음악 소비 트렌드의 변화



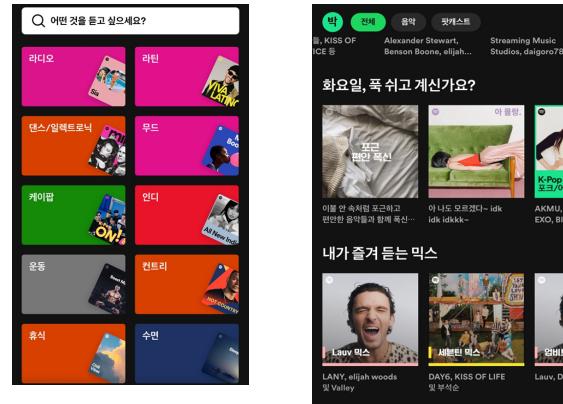
선곡리스트, 유튜브 플레이리스트 등 누군가가 큐레이팅한 플레이리스트를 이용하는 사용자 증가했고 Z세대들이 이러한 플레이리스트를 이용하는 주된 이유는 '내가 모르던 새로운 음

악을 듣기 위함', '상황, 분위기, 테마에 맞는 음악을 듣기 위함'이다.

1.1.3 기존 음악 추천 서비스

Youtube Music, Melon, Spotify, Genie와 같은 점유율 높은 음악 서비스의 경우 모두 기본적으로 아래와 같은 추천 기능을 제공하고 있다.

- 키워드 기반 추천
- 개인화 기반 믹스 플레이리스트
- 개인화, 예상 상황 기반 플레이리스트



이러한 서비스 만으로는 사용자의 다양한 상황과 감정을 반영하기 어렵다는 한계점이 존재한다.

Au-Dionysos는 바로 이러한 부분에서 아쉬움을 느끼는 사람들의 니즈를 충족하기 위한 서비스이다.

1.2 기대효과

- 초개인화 경험 제공

사용자의 현재 감정과 상황을 반영해서 원하는 플레이리스트를 얻을 수 있다.

- 사용자의 시간 절약

기존 음악 스트리밍 서비스에서는 추천해주는 키워드나 믹스, 예상 상황 플레이리스트 이외의 사용자가 원하는 플레이리스트를 제공해주지 않는다. 해당하지 않는 플레이리스트를 원할 때는 직접 음악을 하나하나 골라서 플레이리스트에 추가해야 하는 과정이 있는데, 본 서비스는 이러한 소요 시간을 줄일 수 있다.

- 사용자의 serendipity 실현

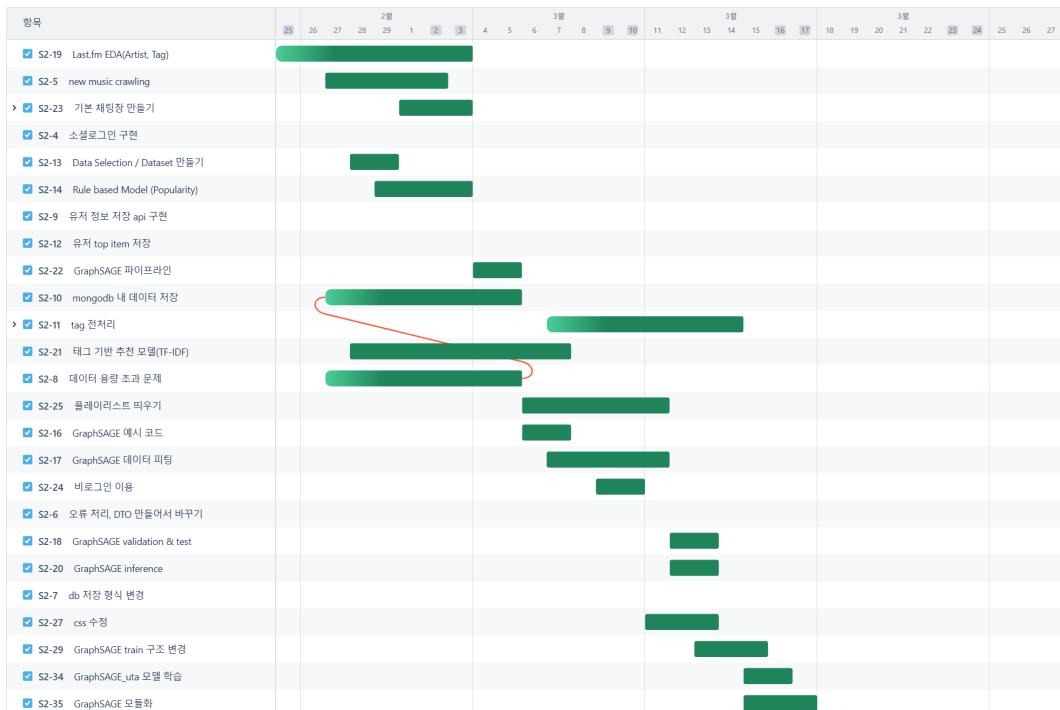
음악 스트리밍 서비스의 주 소비자 계층인 Z세대는 '모르던 새로운 음악을 듣기 위해' 다른 사람이 만든 플레이리스트를 듣는다. 본 서비스는 사용자의 취향을 기반으로 주로 새로운 음악을 추천해준다.

1.3 팀 구성 및 수행 절차

1.3.1 팀 구성 및 역할

	김수빈	프로젝트 기획 LangChain 기반 Chat2Tag 모델 구현 데이터-모델 파이프라인 설계
	이재권	데이터 전처리 추천 시스템 설계 및 구축 GraphSAGE 모델링
	백승빈	프론트엔드 구현 서버 관리 및 서비스 배포 서비스 테스트 구현
	이진민	백엔드 구현 테스트 데이터셋 수집
	박시우	데이터 수집 및 전처리 DB 구축 및 백엔드 구현 UI 태그 추천 로직 구현
	장재원	데이터 전처리 및 임베딩 추천 시스템 설계 및 구축 Content 기반 모델링

1.3.2 프로젝트 수행 타임라인



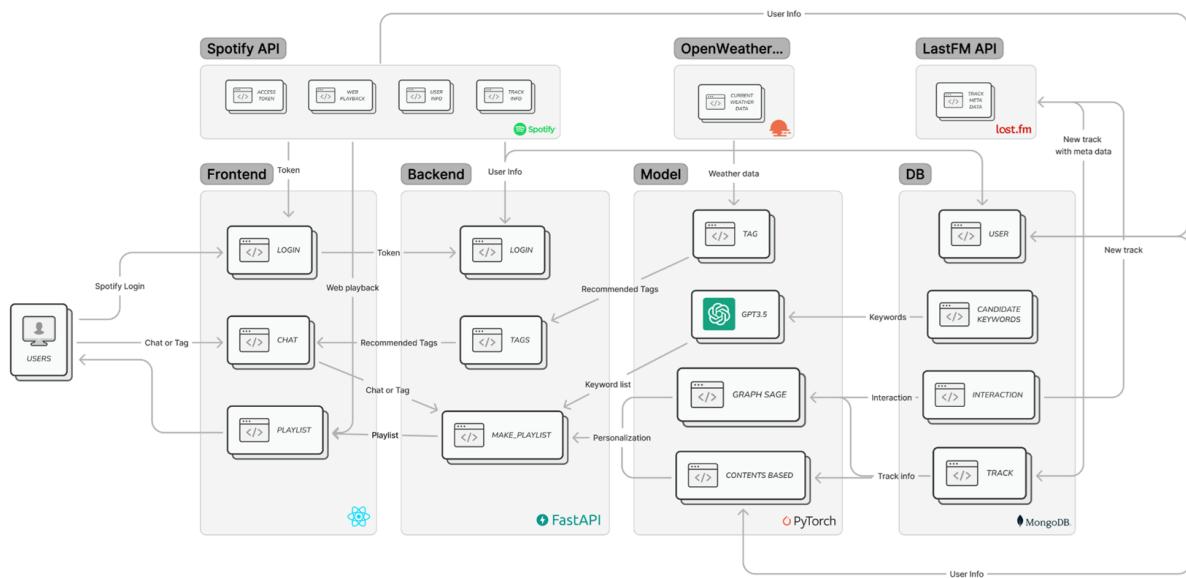
1.4 개발 환경

1.4.1 기술 스택

구분	Skill
Languages	Python, JavaScript
Framework / Library	Pytorch, FastAPI, React
Database	MongoDB

Environment	Linux
ETC	Jira, Notion, Slack

2. 서비스 아키텍처



3. 데이터

3.1 데이터 구성

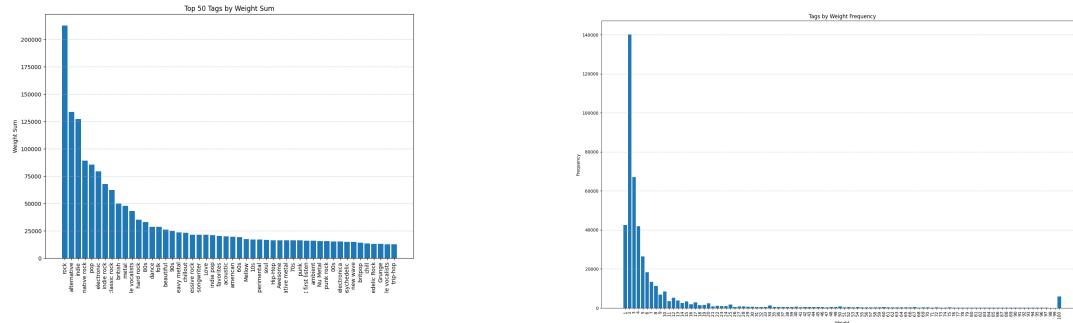
- Lastfm dataset
 - 2020년까지의 spotify 유저의 음악 소비 데이터
 - interaction data, lastfm 유저들이 등록한 곡 태그 리스트
- Spotify dataset
 - 2021.09 ~ 2024.02까지의 신곡 음악 데이터
 - 아티스트 이름, 트랙 이름, 태그 리스트, uri, 12개의 audio features

3.2 데이터 EDA 및 전처리

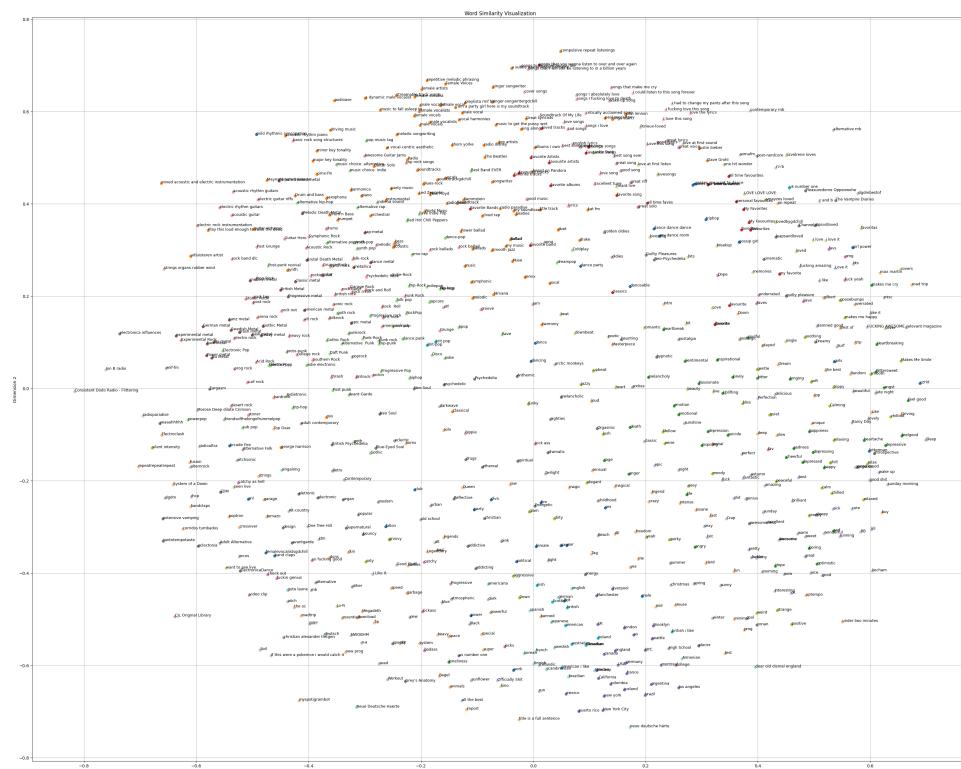
3.2.1 EDA

'Top 50 Tags by Weight Sum' 그래프를 보면 특정 태그들이 빈도가 집중되어 있는 것을 확인할 수 있었다. 이는 몇몇 태그들이 다른 태그들에 비해 상대적으로 더 많이 사용되거나 더 중요하다는 것을 나타낸다. 'Tags by Weight Frequency'그래프에서는 전반적으로 원쪽으로 치우친 분포를 보이며, 동시에 가중치가 100인 태그의 빈도수가 눈에

띄게 높다. 이는 특정 가중치를 갖는 태그들이 매우 자주 등장함을 의미하며, 이는 이중 모드 또는 다봉 분포를 나타내고 있다. 태그 분석에 있어서 의미 있는 정보를 얻기 위해서는 의미가 비슷한 태그들을 그룹화하고, 빈도수가 낮은 태그들을 제거함으로써 데이터의 '꼬리'를 잘라내어 데이터 집중도를 높여야 한다.



Hugging Face에서 제공하는 Sentence Transformer로 고유한 태그 약 52,000개를 임베딩한 후 KMeans Clustering을 진행하여 총 9가지의 대분류 카테고리가 있음을 알 수 있었다. 임베딩 벡터를 2차원으로 mapping한 결과 다음과 같이 어느 정도 군집화가 되어 있었다.



9가지 대분류 카테고리는 다음과 같았다.

- 계절, 하루 중 시간, 발매 날짜
- 감정 및 분위기, 듣기 좋은 상황
- 장르, 가수 정보
- 국가 명, 사용된 언어

3.2.2 전처리

전처리 과정에서 가장 어려웠던 점은 태그가 곡마다 충분히 붙어 있는 게 아니었기 때문에 최대한 유의미한 태그를 보존해야 하면서도 무의미한 태그가 많아 삭제도 많이 해야 하는 trade-off가 있었던 것이다.

즉, 최대한 유의미한 태그를 삭제하지 않고 보존하기 위해 단어 자체의 의미를 인식하면서 전처리를 하기 위해 NER(Named Entity Recognition)을 적용하기로 결정했다. 전처리 단계는 아래와 같았다.

- Sentence Transformer로 임베딩한 값의 유사도를 사용해 같은 의미를 가진 태그를 하나로 묶음
- 곡에 노출된 횟수를 기준으로 Tail 태그 삭제
- Spacy에서 제공하는 모델을 활용해서 NER 조건 적용 및 모델 전이학습을 진행해 유의미한 태그만 보존

보유하고 있지 않은 신곡에 대한 meta 정보에 대한 전처리 파이프라인은 효율성을 위해 1번 단계를 빼고 진행하고자 한다.

이렇게 정제된 태그를 저장하고 난 후 모델에 적용하기 위해 태그 임베딩을 진행했다.

12개의 audio feature의 경우, minmax 정규화를 통해 모든 feature의 값을 0~1 사이의 연속된 값으로 표현하였다.

3.2.3 임베딩

노래를 임베딩하기 위해 다음과 같은 3가지 방법이 사용되었습니다.

1. Tag emotion embedding

목적 : Tag의 감정 상태(Paul Ekman의 6가지 기본감정)를 파악

모델 :

[michellejieli/emotion_text_classifier](#)

2. Tag text embedding

목적 : Tag의 의미적 내용을 파악

모델 :

[jinaai/jina-embeddings-v2-small-en](#)

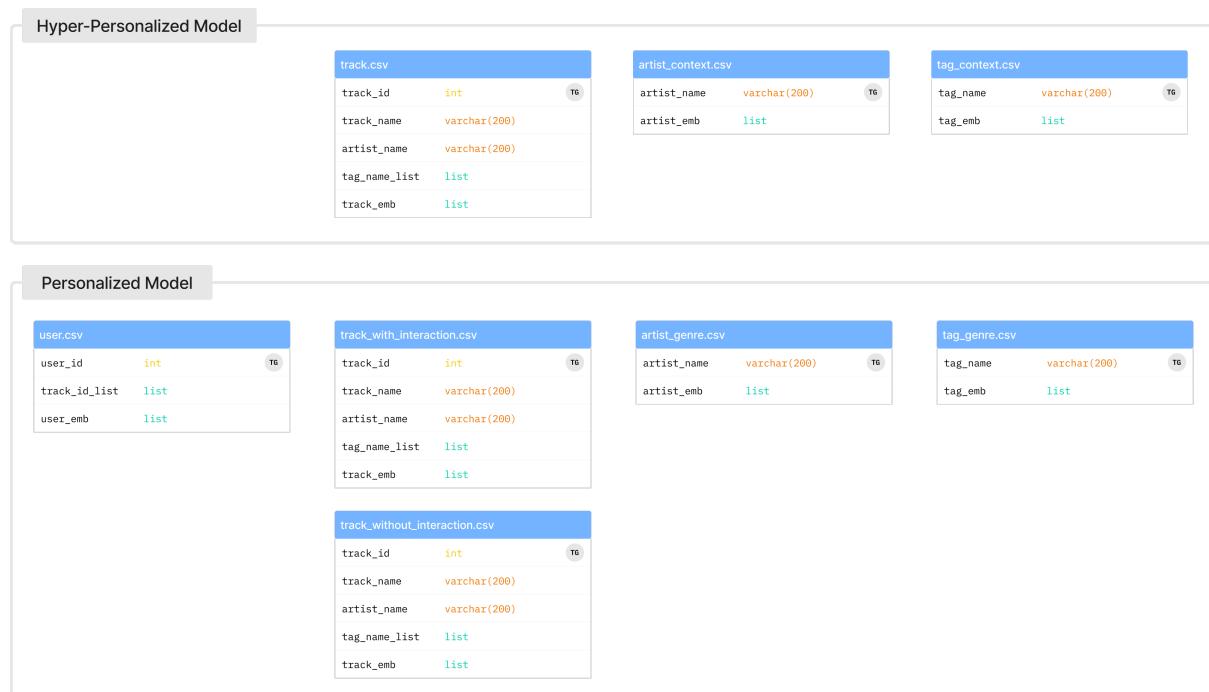
3. Meta data

목적 : 노래의 분위기 파악

모델 : Spotify API

우리는 다음과 같은 가설을 바탕으로 Hyper-Personalized Model과 Personalized Model에 다른 임베딩 값을 사용하였다. 사람의 취향은 감정 및 분위기와 같은 태그들보다 장르가 많은 영향을 미칠 것이고, 현재의 상황을 플레이리스트에 반영하기 위해서는 감정 및 분위기와 같은 태그들이 많은 영향을 미칠 것이다. 그렇기에 Personalized Model에서는 Tag중에서 장르만 사용되어야 한다. 임베딩 순서는 다음과 같다.

1. Tag마다 emotion 및 text embedding을 계산
2. Track에 포함된 Tags를 emotion 및 text embedding의 평균과 Meta data로 track embedding값 설정
3. 각 artist별로 해당되는 Track의 embedding값을 평균내서 artist_emb에 할당
4. 각 tag별로 해당되는 Track의 embedding값을 평균내서 tag_emb에 할당



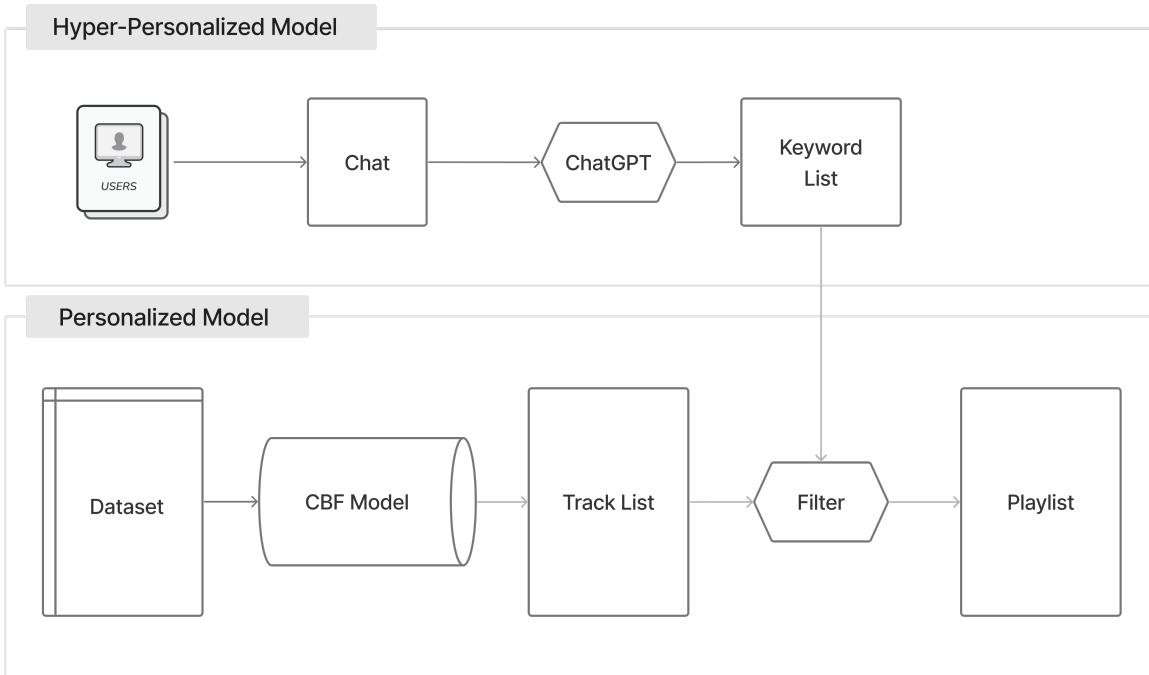
3.3 데이터베이스 구조

Listening_Events		User		Track		Candidate_Keywords	
user_id	integer	user_id	integer	track_id	integer	tag	varchar
track_id	list	email	varchar	artist_name	varchar		
		uri	varchar	track_name	varchar		
		country	varchar	tags	list		
		top_track	list	genres	list		
		new_track	list	uri	varchar		
		tag_counts	dict	danceability	float		
				energy	float		
				key	int		
				loudness	float		
				speechiness	float		
				acousticness	float		
				instrumentalness	float		
				liveness	float		
				valence	float		
				tempo	float		
				duration_ms	int		
				time_signature	int		

데이터베이스 구조는 위와 같았다. User와 Track 컬렉션은 각각 로그인 사용자와 추천해줄 곡에 대해 필요한 정보들을 저장한다. Listening Events는 interaction 데이터를 담고 있고 Candidate Keywords는 유저의 채팅으로부터 추출해낼 만한 고유한 태그를 저장하고 있다.

4. 추천 모델

4.1 ML기반 모델 (1차 배포)



ML기반 모델은 사용자의 과거 상호작용 없이도 유사한 노래를 추천할 수 있는 독립성과, 모델 학습 없이 임베딩 간 비교만으로 추천이 가능하여 온라인 서빙이 용이하다는 장점을 가지고 있다. 그렇기에 1차 배포 모델로 선정하였고 구조는 아래와 같다.

1. Content based filtering model

User가 선호하는 노래들과 Dataset 노래들의 유사도를 계산하여, 유사도가 높은 100개의 노래를 선별하여 Track list를 생성한다.

2. ChatGPT

사용자가 입력한 텍스트에서 Keyword(Tag)를 추출한다. 기존 Track들이 가진 고유의 Tag들을 바탕으로 텍스트와 감정, 상황, 맥락적으로 가장 적합한 태그 5개를 추출한 뒤 반환한다.

3. Playlist 생성

추출된 Keyword 리스트를 기반으로 Track list를 재정렬하여 최종 플레이리스트를 생성한다.

1차 배포 피드백을 받은 결과 다음과 같은 문제점이 발생하였다.

1. 성능 부족 : 사용자의 만족도를 충족시키지 못함
2. 제1종 오류 : 특정 장르나 가수를 입력했을 때 관련 없는 노래가 추천되는 경우가 발생

이를 해결하기 위해서 다음과 같은 부분을 개선하여 2차 모델을 개발하였다.

1. GraphSage 도입: 모델의 성능을 개선하기 위해 GraphSage 알고리즘을 사용
2. 필터 추가: 필터를 추가하여 제1종 오류를 최소화한 추천 실시
3. 임베딩 다양화: 초개인화와 개인화 모델에 서로 다른 임베딩을 사용
4. 데이터셋 전처리 강화: 데이터셋에 대한 전처리를 타이트하게 적용하여 데이터의 품질을 향상

4.2 DL기반 모델 (2차 배포)

GraphSAGE 모델을 기반으로 추천 시스템을 설계하였다. GraphSAGE 모델은 특성 정보의 임베딩을 초기값으로 하고, 노드 간의 관계를 분석하여 노드의 임베딩을 학습한다. 임베딩의 초기값으로 audio feature, 텍스트 임베딩, 텍스트 감성 임베딩을 활용하였고, User-Track, Track-Tag, Track-Artist 관계를 통해 임베딩을 학습하였다.

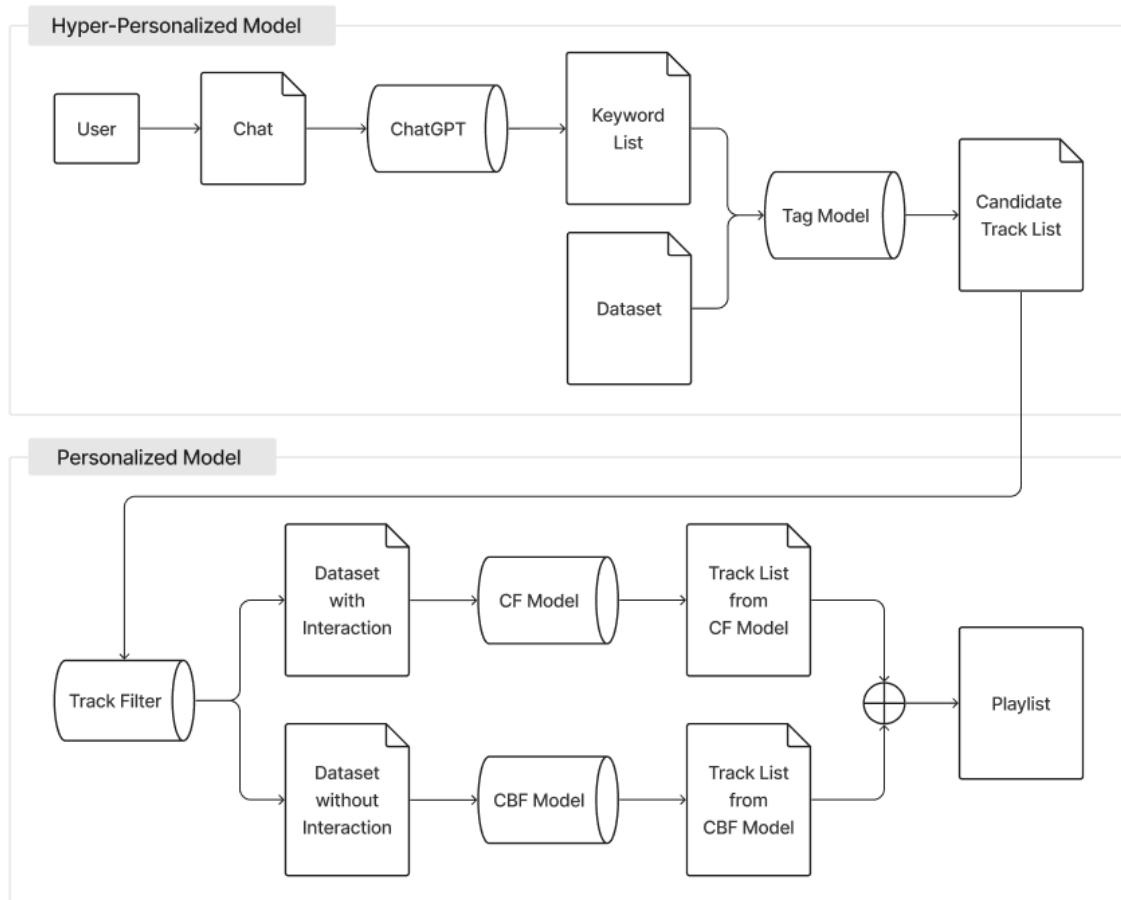
4.2.1 GraphSAGE

정보가 적은 새로운 사용자에게 음악을 추천하는 서비스이기 때문에 정보량의 부족으로 인한 Cold Start 문제가 예상되었다. Cold Start 문제를 완화하기 위해 Interaction 정보와 Meta 정보를 모두 활용하고자 하였다.

Interaction 정보와 Meta 정보를 활용하기 위한 방법으로 GraphSAGE 모델을 선정하였다. GraphSAGE는 그래프 구조를 통해 Interaction 정보와 Meta 정보를 효과적으로 표현할 수 있고, Collaborative Filtering의 특성을 가진 모델과 Content based Filtering의 특성을 가진 모델을 생성할 수 있다.

GraphSAGE는 그래프 구조를 통해 이웃 노드에 정보를 전파하고, 이웃 노드의 정보를 통합하여 노드의 특성을 표현하는 학습 방식을 사용한다. 이 방법은 기존 데이터에 없는 새로운 노드를 학습하는 데에 유리하다.

4.2.2 추천 시스템 파이프라인



1. ChatGPT

사용자가 입력한 텍스트에서 Keyword(Tag)를 추출한다. 기존 Track들이 가진 고유의 Tag들을 바탕으로 텍스트와 감정, 상황, 맥락적으로 가장 적합한 태그 5개를 추출한 뒤 반환한다.

2. Tag Model

Track과 Tag의 관계를 GraphSAGE 구조로 학습한다. 입력된 Tag와 관련된 Track을 추천한다.

3. Track Filter

User-Track interaction의 유무로 데이터를 나눈다. LastFM 데이터에는 Interaction 정보가 존재하지만, Spotify 데이터에는 Interaction 정보가 존재하지 않기 때문에 두 가지 모델로 나누어 학습한다.

4. CF Model

User의 청취 기록을 바탕으로 User 임베딩을 생성한다. User와 Track의 관계를 GraphSAGE 구조로 학습하고, Track과 Genre Tag의 관계를 보조적으로 활용한다. 입력된 User가 좋아할만한 Track을 추천한다.

5. CBF Model

Track과 Genre Tag의 관계를 GraphSAGE 구조로 학습한다. 입력된 Track과 유사한 Track을 추천한다.

6. Shuffle

CBF Model의 결과와 CF Model의 결과를 하나씩 번갈아 출력하는 방식으로 Shuffle한다.

4.3 UI Tag 모델

기존 음악 스트리밍 서비스에서는 개인화된 태그 목록을 추천해주진 않는다. 유튜브에서는 사용자의 영상 소비 목록을 기반으로 키워드를 추천해주는 것이 사용자 입장에서 서비스에 대한 좋은 인상을 남길 수 있다는 생각으로 사용자의 interaction 데이터와 날짜, 시간, 날씨를 기반으로 개인화된 태그를 추천해주고자 하였다. 로그인 사용자가 아닌 경우 interaction 데이터가 없으므로 고정된 태그 목록과 날짜, 시간, 날씨만을 기반으로 태그를 추천해주었다.

- 사용자의 interaction 데이터 사용

사용자가 소비한 음악의 목록이 적을 경우 취향을 판단하기 어렵다고 생각해 5곡 이상을 소비한 경우에 소비한 곡의 태그 목록 중 많았던 태그를 추천했다.

- 날짜 데이터 사용

3~5월인 경우 봄, 6~8월인 경우 여름, 9~11월인 경우 가을, 12~2월인 경우 겨울로 판단하여 그에 맞는 계절 태그를 추천했다.

- 시간 데이터 사용

아침 시간대와 저녁 시간대에 맞는 태그를 추천했다.

- 날씨 데이터 사용

비가 오거나 눈이 오는 날씨는 사람들이 듣는 음악에도 영향을 많이 미칠 것이라고 생각해서 그러한 경우에 적합한 태그를 추천했다.

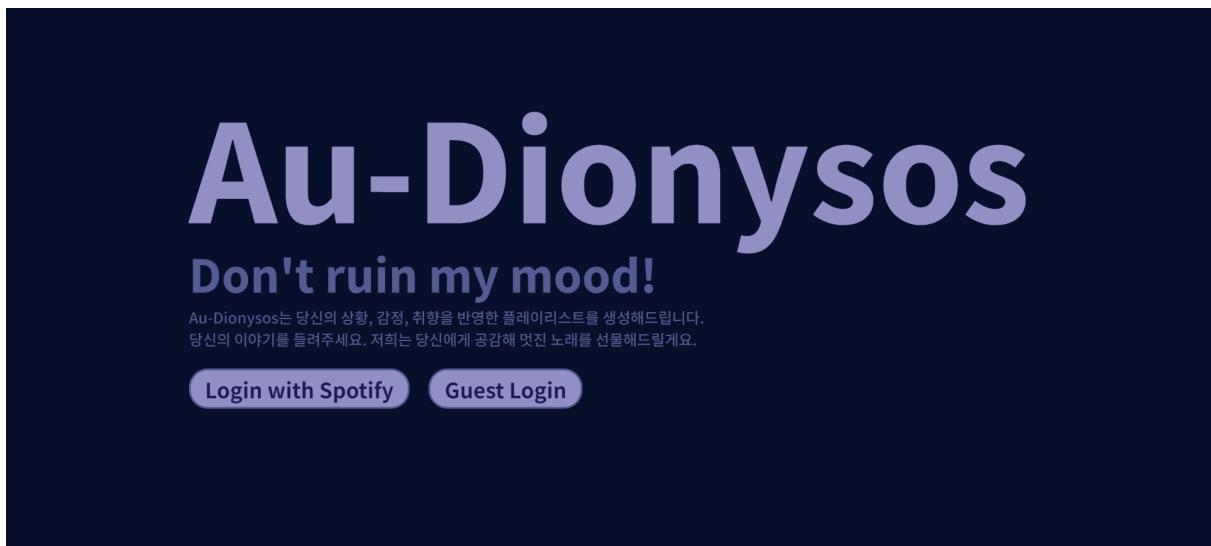
5. 서비스 배포

유저들에게 원활한 서비스를 제공하기 위해 도메인을 구매하고 웹 서버와 연결하였다. 프론트엔드 서버와 백엔드가 다른 서버를 가지고 있기 때문에 nginx를 이용해 리버스 프록시를 구현하여 도메인에서 들어오는 요청들을 적절히 포워딩되도록 했다. 또한 1차 배포에서 다수

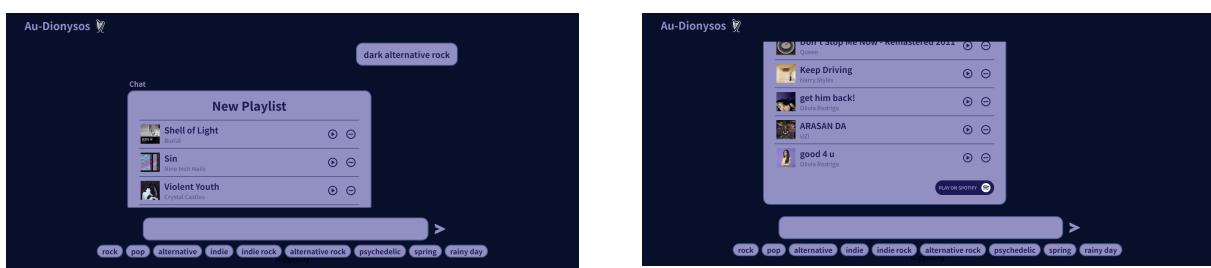
의 유저가 접속할 경우 속도가 급격히 느려진다는 점을 발견하고 도메인에 추가적인 IP를 연결함으로써 DNS 라운드 로빈을 구현하여 문제를 완화했다.

6. 서비스 화면

6.1 로그인 화면

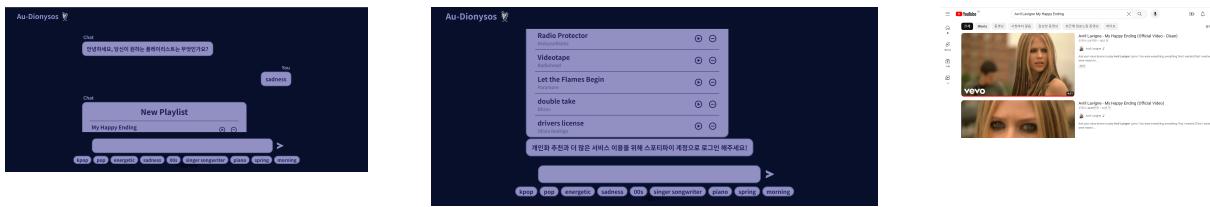


6.2 스포티파이 로그인 유저



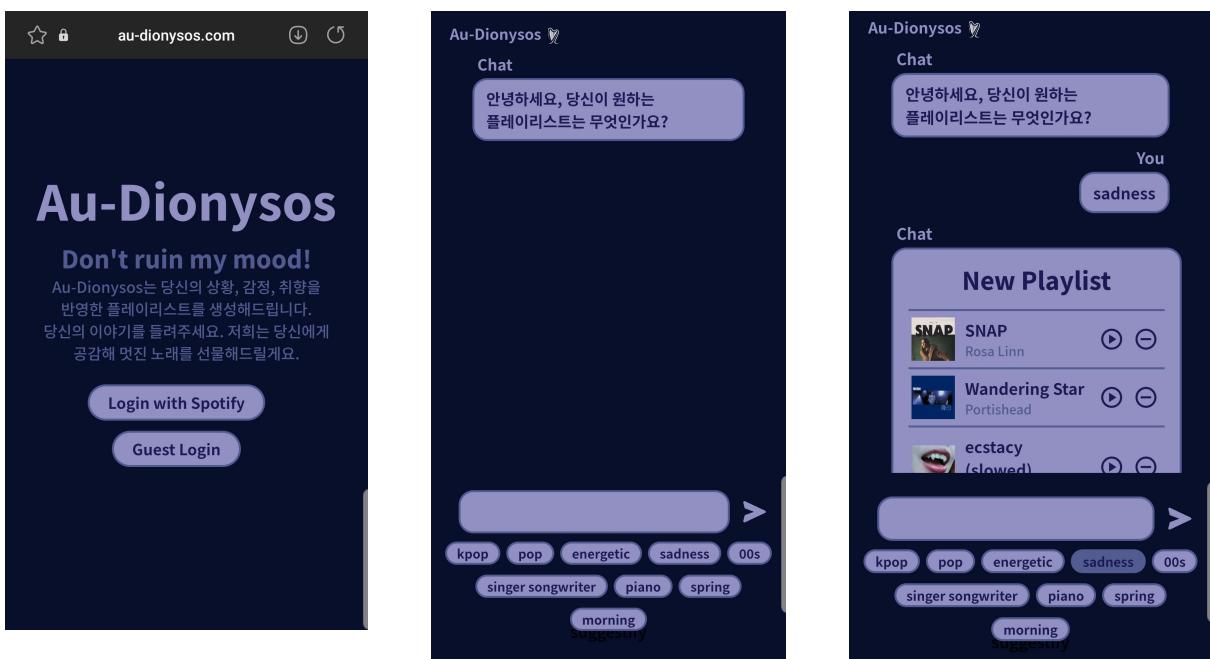
유저는 채팅 또는 하단의 태그를 선택하여 플레이리스트를 추천 받을 수 있다. 추천된 플레이리스트에서 노래를 재생해보고 마음에 들지 않는 곡은 목록에서 삭제할 수 있다. 또한 Play on Spotify 버튼을 클릭하여 스포티파이에 해당 플레이리스트를 저장할 수 있다.

6.3 비로그인 유저



스포티파이 계정을 가지고 있지 않은 유저는 개인화되지않은 추천을 제한된 기능과 함께 제공받을 수 있다. 플레이 버튼을 누르면 웹에서 재생되는 대신 유튜브 화면으로 연결된다.

6.4 모바일 화면



반응형 웹을 구현하여 유저들이 모바일기기로도 편리하게 접근할 수 있도록 하였다.

7. 결과 및 고찰

7.1 결과

7.1.1 모델 평가

NDCG@100 (CF 모델 : User-Track 관계로 평가)

	Collaborative Filtering	GraphSAGE CF Model
Train	0.0532	0.2210

	Collaborative Filtering	GraphSAGE CF Model
Valid	0.0546	0.2050
Test	0.0532	0.2032

NDCG@100 (CBF 모델 : Tag-Track 관계로 평가)

	Content based Filtering	GraphSAGE CBF Model
Train	0.0316	0.3370
Valid	0.0113	0.1641
Test	0.0102	0.1636

CF 모델은 User-Track 관계를 정답으로 설정하여 NDCG@100으로 모델을 평가하였고, CF 모델은 Tag-Track 관계를 정답으로 설정하여 NDCG@100으로 모델을 평가하였다.

특성 정보를 그대로 활용한 CF 모델, CBF 모델과 비교하여 GraphSAGE 기반 모델이 특성 정보와 노드 간의 관계를 학습하여 더 우수한 추천 성능을 보였다.

7.1.2 설문조사 결과

서비스를 외부에 배포해 사용자들의 만족도를 파악해보고 개선할 점이 무엇인지 알아보고자 하였다.

1차, 2차로 두 번에 걸쳐 배포를 함으로써 사용자의 피드백을 기반으로 개선했을 때 어느 정도로 만족도가 높아질 것인지 평가해보고자 하였다.

1차 배포 설문조사 참여자는 총 27명으로 결과는 아래와 같았다.

- 스포티파이 로그인 사용자 3명, 비로그인 사용자 24명이 참여했다.

로그인 사용자의 피드백을 분석하기에는 표본이 너무 적어 비로그인 사용자에 대한 만족도를 아래와 같이 정리했다.

- 채팅창 아래의 추천 키워드 적절성 여부에 대한 질문에 매우 그렇다(1), 그렇다(11), 보통이다(9), 그렇지 않다(2), 매우 그렇지 않다(1)로 긍정적인 답변이 50% 였다.
- 키워드 선택으로 추천받은 경우 플레이리스트가 잘 추천되었는지에 대한 질문에 매우 그렇다(2), 그렇다(8), 보통이다(6), 그렇지 않다(3), 매우 그렇지 않다(2)로 긍정적 답변이 48%였다.
- 채팅으로 추천받은 경우 플레이리스트가 잘 추천되었는지에 대한 질문에 매우 그렇다(1), 그렇다(7), 보통이다(8), 그렇지 않다(2), 매우 그렇지 않다(6)로 긍정적 답변이 33%였다.

- 개인화된 추천과 웹 내 원활한 음악 청취를 위해 스포티파이 계정을 통한 서비스 이용을 해볼 용의가 있느냐라는 질문에 있다(17), 없다(7)로 긍정적 답변이 71%였다.
- 서비스에 대한 전반적 만족도는 매우 그렇다(1), 그렇다(10), 보통이다(10), 그렇지 않다(6), 매우 그렇지 않다(0)로 긍정적 답변이 41%였다.

2차 배포 설문조사 참여자는 총 25명으로 결과는 아래와 같았다.

- 스포티파이 로그인 사용자 6명, 비로그인 사용자 19명이 참여했다.

로그인 사용자의 피드백을 분석하기에는 표본이 너무 적어 비로그인 사용자의 만족도를 아래와 같이 정리했다.

- 채팅창 아래의 추천 키워드 적절성 여부에 대한 질문에 매우 그렇다(4), 그렇다(12), 보통이다(2), 그렇지 않다(0), 매우 그렇지 않다(0)로 긍정적인 답변이 89%였다.
- 키워드 선택으로 추천받은 경우 플레이리스트가 잘 추천되었는지에 대한 질문에 매우 그렇다(6), 그렇다(8), 보통이다(4), 그렇지 않다(0), 매우 그렇지 않다(0)로 긍정적 답변이 77%였다.
- 채팅으로 추천받은 경우 플레이리스트가 잘 추천되었는지에 대한 질문에 매우 그렇다(4), 그렇다(8), 보통이다(2), 그렇지 않다(2), 매우 그렇지 않다(0)로 긍정적 답변이 75%였다.
- 개인화된 추천과 웹 내 원활한 음악 청취를 위해 스포티파이 계정을 통한 서비스 이용을 해볼 용의가 있느냐라는 질문에 있다(12), 없다(6)로 긍정적 답변이 67%였다.
- 1차 배포 서비스 경험자는 총 13명으로, 1차 배포와 비교 시 추천 기능이 향상되었다고 느끼는지에 대한 질문에 매우 그렇다(3), 그렇다(9), 보통이다(1)로 긍정적 답변이 92%였다.
- 서비스를 다시 이용할 용의가 있느냐는 질문에 있다(20), 없다(2)로 긍정적 답변이 91%였다.
- 서비스에 대한 전반적 만족도는 매우 그렇다(4), 그렇다(13), 보통이다(5), 그렇지 않다(0), 매우 그렇지 않다(0)로 긍정적 답변이 77%였다.

대부분의 질문에서 사용자의 긍정적인 답변이 크게 증가한 것을 확인할 수 있었다.

7.2 고찰

7.2.1 개선점 및 미래 방향

- 모델의 경량화

현재 총 latency 약 2.75초 중 모델 로딩에 2초 가량 소요된다. 추천이 필요할 때마다 모델을 로딩하는 현재의 방식은 비효율적이라는 지적이 있었다. 이를 반영해 서버를 올릴 때 모델을 저장해두는 방식을 통해 latency를 줄일 계획이다.

- 모델 재학습 자동화

새로운 신곡 정보를 업데이트하고 모델을 재학습시키는 파이프라인이 아직 구현되어 있지 않다. 지속적인 서비스 사용을 위해서 이 부분을 구현할 계획이다.

- 프롬프트 오류 핸들링

프롬프트에 예상치 못한 문장이 들어왔을 때 주어진 형태가 아닌 다른 형식의 답변을 주는 경우가 있다. 프롬프트 엔지니어링만으로는 이 문제를 해결하기 어렵다는 판단 하에 이를 처리하기 위한 기능을 추가할 계획이다.

8. 개인 회고

8.1 김수빈

이번 프로젝트는 기획의도부터 '초개인화'를 상정하고 만들었다. 나의 이동경로를 추적해 이동 경로에 맞는 상품 및 서비스를 추천하거나, 나의 행동 데이터를 통해 나에게 딱 맞는 콘텐츠를 추천하는 등 앞으로 모든 산업 및 도메인에서 '초개인화'라는 키워드가 점차 떠오를 것이라고 생각했기 때문이다. 이 프로젝트는 이러한 초개인화를 가상으로 시뮬레이션한 프로젝트에 가깝다.

이번 프로젝트를 진행하며 시도한 것

내가 이번에 맡은 역할은 데이터-모델 간의 파이프라인을 설계하고, Langchain을 통한 ChatGPT 기반 어플리케이션을 개발하는 일이었다. 거기에 더하면 유저로부터 Implicit FeedBack을 받아 이를 재학습해 모델과 데이터를 업데이트하는 것이 마지막 목표였다. 다만 모델의 개발이 늦어지면서, 내가 했던 일들의 과정이 늦어진 문제가 있었다. 초기에 구상했던 대로 CF나 MF 등 간단한 모델로 BaseLine을 설계해서 빠르게 개발을 이어나갔다면, 내가 개발할 시간을 조금 더 확보할 수 있지 않았을까 하는 아쉬움이 있다.

나의 역할은 새로운 일을 늘이기보다, 기존의 일을 확인하고 새로운 일이 커지지 않도록 제한하는데 역할을 다했다. 시간을 맞춰서 개발을 해야 할 필요가 있었기에 이런 역할은 욕을 먹더라도, 꼭 필요했으리라고 생각했다.

소프트웨어공학의 수많은 내용들을 차용해 프로젝트의 진행을 관리했다. 도구 및 기술 스택을 선정하면서도 왜 그 도구를 선택했니? 하는 질문으로 시작해 도구를 선택하고자 했다. 백엔드 프레임워크로는 무거운 Django 대신 가볍고 빠른 개발이 가능한 FastAPI, 상대적으로 힘이 덜 요구되는 텍스트 모델에 LLM 모델을 직접 만들기보다는 빠른 배포에 유리한

ChatGPT + Langchain을 선택한 것이 그 예시 중 하나이다. 물론 추후에는 직접 LLM 모델을 설계할 필요성을 느끼고 있지만, 현재로서는 그 선택들이 적합했다고 생각했다.

다음 프로젝트에서 시도해 볼 것

추후 해볼만한 점은, Ver.1이 출시되었으니 이를 GitFlow를 활용해 Main, Develop, Feature, Release, Hotfix Branch로 나누어 작업하는 것이다. Main Branch 하나만 제대로 만드는 trunk-based 방식은 지속적으로 Main Branch의 수정이 일어나서, 오류가 발생했을 때 과거의 어느 시점으로 Roll-Back을 하기가 어려웠던 것이다. 그래서 이런 개발방식을 도입할 때는 테스트를 자동화해 지속적으로 테스트를 진행해야함을 알았다. 어쨌든 GitFlow의 5개의 Branch로 관리하면, 조금 더 프로젝트의 관리가 쉬워질 것이라고 생각했다.

그리고 현재는 Airflow를 활용한 학습 자동화 파이프라인을 설계함과 동시에 DB로 데이터를 모두 이관하는 작업을 수행중이다. 현재의 모델은 성능 측면에서는 좋으나, 파생되는 데이터가 많고, 모델의 아키텍쳐도 상당히 복잡한 편이다. 이러한 요소들을 핸들링하고, 데이터의 품질을 일관화시키는 것이 수많은 노력이 들어간다는 것을 알 수 있었다. 아직은 진척이 느리지만, 4월 16일 전까지 모두 마무리해서 완성시키는 것을 목표로 하고 있다.

8.2 박시우

이번 프로젝트를 진행하며 시도한 것

- 자연어 전처리 자동화 파이프라인 설계

이번 프로젝트동안 사용되었던 주요 데이터인 태그는 자연어 데이터로 다양한 노이즈가 있는 데이터였다. 이러한 노이즈를 없애면서도 데이터를 보존해야 했지만 약 52,000개의 태그를 직접 다 전처리할 수는 없었기도 하고 실제 서비스라면 전처리도 자동화해야 한다는 생각에 최대한 자동화를 해서 진행하려고 전처리 파이프라인을 설계해서 진행했다.

이번 프로젝트를 진행하며 아쉬웠던 것

- 전처리 자동화 파이프라인 구현

앞서 말했듯이 설계는 이루어졌지만 구현은 다 이루어지지 못했다. 아쉬움이 남아서 이 부분은 추가 구현 예정이다.

- 실제 대상 유저에 테스트

한국에서 스포티파이를 사용하는 사람은 현저히 적어서 2번 배포 때 모두 실제 회원 사용자로부터 유의미한 피드백을 얻어내지 못했고 결과적으로 비회원 대상자의 피드백 영향이 너무 컸다. 외국 유저를 대상으로 피드백을 얻으려는 시도를 했다면 성능 테스트에 더 좋았을 것 같다는 아쉬움이 든다.

다음 프로젝트에서 시도해 볼 것

- 서비스 아키텍처 설계

서비스 아키텍처를 설계하고 진행하긴 했지만 그 과정에서 tool 선정 이유가 미흡했다는 생각이 많이 들었다. MongoDB 사용에 대한 지적을 여러 번 듣기도 했었다. 다음 프로젝트를 하게 된다면 서비스 아키텍처를 A부터 Z까지 직접 구상해보는 경험을 갖고 싶다.

8.3 백승빈

이번 프로젝트를 진행하며 시도한 것

- 프론트 엔드

프로젝트의 프론트엔드 부분의 전반을 맡아 진행을 했었다. 음악의 정보를 가져오거나 재생을 하기위해 Spotify API를 많이 사용하게 되었는데 Spotify API를 사용하게 된 것보다는 공식 document를 보며 여러 기능들을 파악하고 응용할 수 있는 능력을 기른 것이 더 큰 수확인 것 같다. AI 개발 프로젝트이긴 했지만 최종 목표가 실제 유저가 사용 할 수 있는 서비스 개발이었던 만큼 음악 삭제, Spotify 연결, 비회원 음악 청취 등 유저의 편의성을 위한 기능을 추가하고자 했다.

- 서비스 배포

유저들에게 원활한 서비스를 제공하기 위해 도메인을 연결하고 웹 서버를 구축했다. 백 엔드 서버와 프론트 엔드 서버를 묶기 위해 nginx를 사용하여 리버스 프록시를 구현했다. 또한 1차배포 후 다수의 유저 접속 시 서비스 속도가 급격히 느려지는 것을 보고 빠른 시간 내 속도를 개선하기 위해 DNS 라운드 로빈을 사용했다. 진행을 하며 많이 헤맸지만 네트워크와 서버의 구조에 대한 공부가 많이 되었던 것 같다.

이번 프로젝트를 진행하며 아쉬웠던 것

- 비로그인 유저

이 프로젝트는 Spotify 유저를 대상으로 하는 것이라 상정하고 시작했기 때문에 비로그인 기능은 정말 추가적인 것이었다. 하지만 배포 때 대부분의 유저가 비로그인 기능을 사용하며 본 서비스에 대한 피드백은 듣지 못했던 것 같다. Spotify API 사용이 필요한 기능들은 제외하더라도 직접 좋아하는 곡들을 입력하는 방식으로 비로그인 유저에게도 개인화 추천이 가능하게 했다면 어땠을까하는 아쉬움이 남는다.

다음 프로젝트에 시도해 볼 것

- 목적에 맞는 브랜치 관리 전략

이번 프로젝트도 지난 대회와 유사하게 github플로우로 진행했는데 서비스 개발 프로젝트이기 때문에 배포하거나 아니면 문제가 생겨 롤백이 필요할 때 불편이 있었다. 다음에 서비스 개발 프로젝트를 진행한다면 github flow를 사용해 배포 단위로 브랜치를 관리하면 좋을 것 같다.

8.4 이재권

이번 프로젝트를 진행하며 시도한 것

- GraphSAGE 모델 구현

GraphSAGE 모델은 Interaction 정보와 Meta 정보를 함께 학습할 수 있고, 새로운 노드를 학습하는 데에 유리하기 때문에 GraphSAGE 모델을 선정하여 추천 시스템을 구현하였다. 그 과정에서 PyTorch Geometric 라이브러리의 공식 문서를 참고하여 모델 파이프라인을 직접 설계하고, 코드를 구현하였다.

- Hybrid 모델

개발 초기에는 CF의 특징과 CBF의 특징이 모두 반영된 Hybrid 모델을 만들었다. 하지만 User-Track 정보는 Track-Tag 정보에 비해 약 50배 더 많은 정보를 가지고 있었다. 이로 인해 User-Track 정보 위주로 학습되어 LastFM 데이터에 포함된 과거에 발매된 외국 음악을 중심으로 추천이 행해졌다. 이러한 문제를 해결하기 위해 LastFM 데이터는 CF 모델로, Spotify 데이터는 CBF 모델로 학습하였다.

이번 프로젝트를 진행하며 아쉬웠던 것

- 추천 성능

offline 평가 결과에 비해 실제 사용자의 만족도는 높지 않았다. 데이터의 품질 문제도 있었고, 데이터와 더 잘 맞는 모델 파이프라인을 설계했어야 했다. 데이터에 모델을 맞추다 보니 복잡한 파이프라인을 설계하게 되었는데, 서비스 관점에서 더 단순하고 성능이 좋은 파이프라인이 필요했다. 데이터를 철저하게 분석하고, 알맞은 데이터를 추가로 수집한 후에 모델링을 진행하는 것이 더 좋은 방법이었을 것 같다.

- 데이터 품질

LastFM 데이터와 Spotify 데이터의 성질이 달랐다. LastFM 데이터는 과거에 발매된 음악으로 구성되어있어 Tag의 개수가 많았고, 사용자 청취 정보가 존재했다. 반면에 Spotify 데이터는 최근에 발매된 음악으로 구성되어있어 Tag의 개수가 적었고, 사용자 청취 정보가 존재하지 않았다. 이로 인해 LastFM 데이터는 CF 모델로, Spotify 데이터는 CBF 모델로 학습하였는데, 데이터의 성질이 비슷했다면 Hybrid 모델로 만들 수 있었을 것이다.

다음 프로젝트에 시도해 볼 것

- 완성도 있는 서비스를 위한 많은 데이터 확보

사용자의 니즈를 충족시킬 수 있을 만큼 많은 데이터를 사용하지 못했다. 사용자는 텍스트를 통해 구체적인 요청을 입력하는데 비해 9420개의 음악 데이터와 790개의 Tag 데이터로는 사용자의 구체적인 요청에 적합한 추천을 제공하기에 부족했던 것 같다.

- GPT 기반 태그 생성 모델

LastFM의 Tag 품질이 좋지 않았고, LastFM 데이터와 Spotify 데이터의 Tag 종류, 개수에 차이가 있어서 Tag Model의 성능이 만족스럽지 못했다. 차후에는 GPT 기반 태그 생성 모델을 제작하고, 이를 활용한 Tag 품질 향상과 Tag 개수 증가를 통해 더 좋은 모델을 완성하고 싶다.

8.5 장재원

이번 프로젝트를 진행하며 시도한 것

- ML production

서비스가 런칭되기 위해서는 Data, Modeling, Backend, Frontend가 순서대로 작업이 이루어진다. Backend와 Frontend는 Modeling이 진행된 이후에 진행될수 있는 문제가 있다. 이를 해결하기위해 ML기반 모델을 빠르게 구현하여 다른팀들도 작업이 진행될수 있게 하였다.

- 1차배포 피드백을 반영한 2차배포

1차 배포 이후, 사용자로부터 다수의 부정적인 피드백을 받았습니다. 이를 해결하기 위해 임베딩 다양화, GraphSage 및 Filtering 도입과 같은 방법으로 서비스를 개선하였다.

이번 프로젝트를 진행하며 아쉬웠던 것

- Baseline 및 Release의 비체계화

프로젝트 초기 단계에서 ML 기반의 Baseline을 신속하게 구축하고 모델링 파트에서 모델을 발전시켰다. 그러나 모델 구조와 데이터 구조가 변경될 때마다 백엔드에 반영하는 과정이 너무 잦은 수정을 요구했다. 이는 프로젝트의 효율성을 저하시키는 주요 원인이 되었다. 크고 작은 수정 사항에 대한 명확한 기준을 설정하고, 다른 팀과는 큰 변경 사항이 있을 때만 협업을 진행했어야 했던 것 같다.

- ML model에서의 방법을 DL에 빠르게 적용하지 못한것

서비스가 마주한 주요 문제 중 하나는 데이터셋의 불균형이었다. LastFM 데이터셋은 노래와 태그가 많은 반면, Spotify API 데이터셋은 노래와 태그가 적었다. ML 모델에서 이러한 한계를 인지하고 데이터 전처리 및 필터링을 적용했음에도 불구하고, DL 모델을 배포할 때 이러한 문제점들을 해결해줄 것이라는 잘못된 가정 하에 데이터 전처리 및 필터링을 신속하게 적용하지 않았던 점이 아쉽다.