Soundalike

Technology: Recommender System

Group 14: Robert, Ajay, Maria

Simply put, Soundalike recommends songs

01 Using the Million Song Challenge Dataset, we created a recommender system to create a curated playlist for any user.

02 Our final product is a React web app with a Flask server that allows users to input a song and get a list of recommended songs.

01

Motivation & Goal

Motivation

Everyone's journey with music is different and unique -- and in 2022, technology should be able to curate music for you and you only.

Our Goal

We aren’t trying to reinvent the wheel.

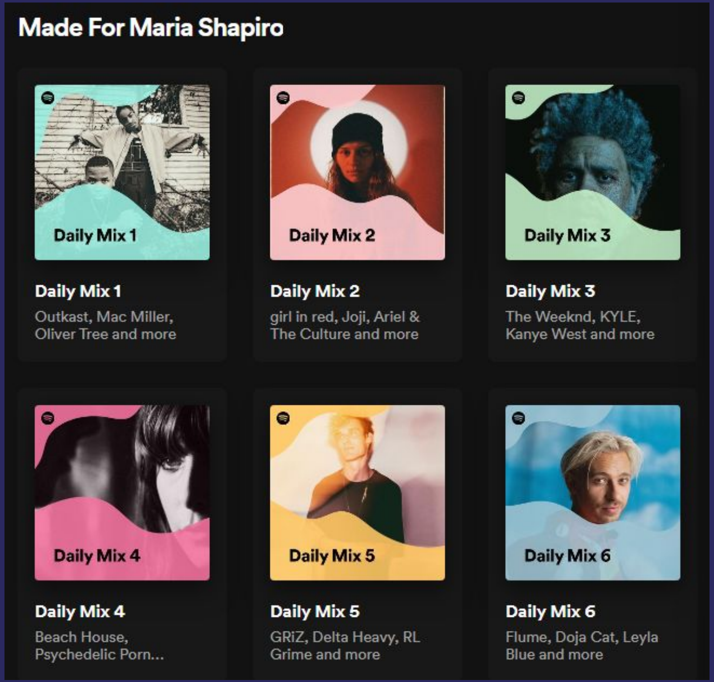
- Spotify, Apple Music, and Youtube Music already have massive user bases.
- These streaming giants have much more time, funding, and data than we do.
- We offer a good idea to fill in the gaps that these streaming platforms have.

02

Inspirations

01 Spotify

Spotify has 180 million paying subscribers, making it the most-subscribed music service. They have a suite of curated playlists called Daily Mix.

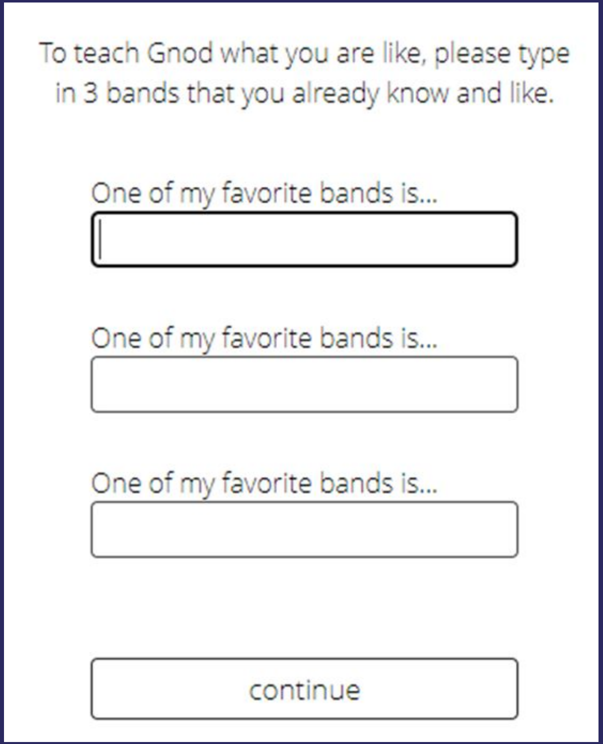


Weaknesses:

- Little transparency on how songs are chosen
- Doesn't often show new music and artists
- Doesn't vary much from day to day

02 Gnoosic

Gnoosic has about 200,000 visitors/month. As a smaller service, it does not include a music player.



Weaknesses:

- Recommends artists, not songs
- No integration with any music services

03

App Functionality

Dataset -- Million Song Dataset Challenge

Open-access with listening history for 1 million+ users

MSD Challenge Data Components

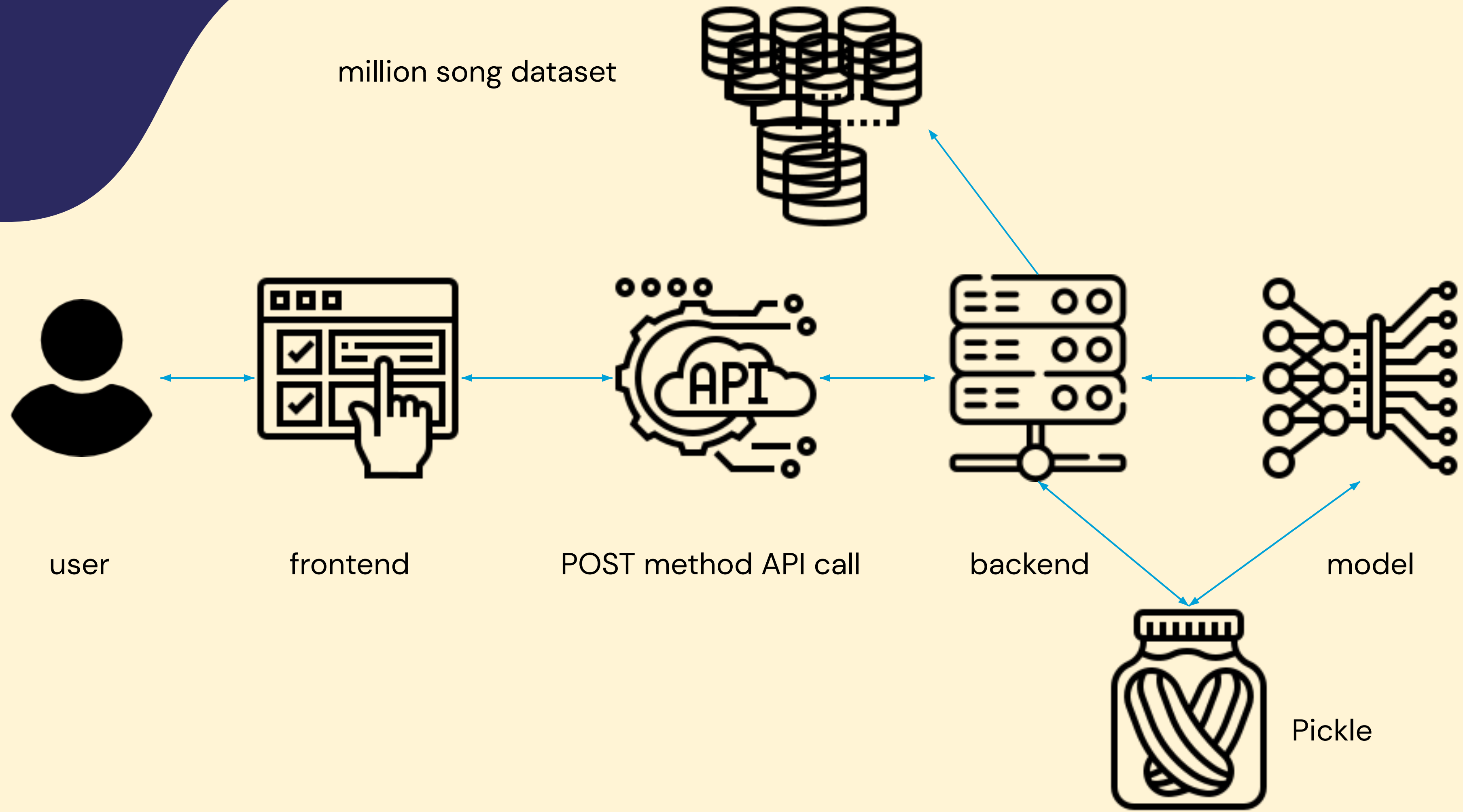
- Listening triplets - user-song pairs with listen count
  - Used to construct user-song sparse implicit feedback matrix
  - Split into train and test splits
- Mapping of song ID to song title
  - Find song ID from user input (handle error if not found)
  - Use recommended song IDs to look up titles

User-Song Listening Triplets

User ID	Song ID	Listens
fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOBONKR12A58A7A7E0	1
fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOEGIYH12A6D4FC0E3	1
fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOFLJQZ12A6D4FADA6	1
fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SOHTKMO12AB01843B0	1
fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	SODQZCY12A6D4F9D11	1

Song ID to Title Mapping

Track ID	Song ID	Song Title	Artist Name
TRMMMYQ128F932D901	SOQMMHC12AB0180CB8	Faster Pussy cat	Silent Night
TRMMMKD128F425225D	SOVFVAK12A8C1350D9	Karkkiautomaatti	Tanssi vaan
TRMMMRX128F93187D9	SOGTUKN12AB017F4F1	Hudson Mohawke	No One Could Ever
TRMMMCH128F425532C	SOBNYVR12A8C13558C	Yerba Brava	Si Vos Querés
TRMMMWA128F426B589	SOHSBXH12A8C13B0DF	Der Mystic	Tangle of Aspens


million song dataset
user
frontend
POST method API call
backend
model
Pickle

Frontend

For the front end, we are utilizing React and deploying with Vercel

We chose this for a variety of positives:

- Mix of HTML, CSS, and JavaScript in JSX which makes React incredibly flexible
- Virtual DOM (Document Object Model) is responsible for React's performance and speed
- Library's strong community support - can be attributed to the fact that React is open-source and includes many packages to aid development
- Used ant-design library for React components

Backend

We are using Python Flask as our backend server because it is quick to connect and has very minimal boilerplate code. Our entire team is very familiar with Python.

- Service layer handles POST request call (axios) from React frontend
- Handler class validates inputted song title with Million Song dataset using Numpy and Pandas
- Sends song title to ML model
- Retrieves recommended songs from ML model to display on the frontend

Open Source Modifications

- Upgraded Python to 3.9 and Pandas to 1.4.2 to ensure pickle.load() worked
  - Machine that performs pickle.dump() needs version of Pandas compatible with the machine that performs pickle.load()
- Added CORS preflight headers to handle CORS errors when React makes POST requests to Flask

Matrix Factorization Models

1. Logistic Matrix Factorization (LMF)
2. Alternating Least Squares (ALS)

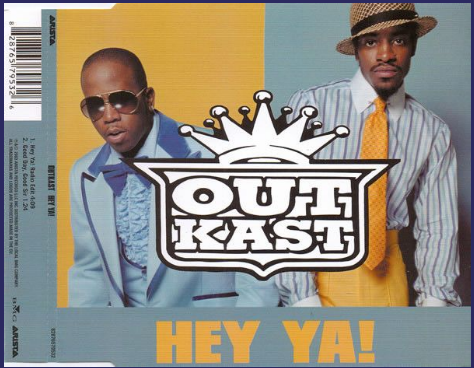
Recommendation Quality

- Precision-at-*k* metric:

$$P_k(u,y) = \frac{1}{k}\sum_{j=1}^{k} Mu,y(j)$$

- Proportion of top-*k* predicted rankings a user listened to in testing dataset

Song Recommendation Example



Hey Ya! – Outkast

1. Please Please Please – Shout Out Louds
2. Tiny Explosions – The Presidents of the United States of America
3. Already Gone – Kelly Clarkson
4. The Way You Lived – CKY
5. Greenback Dollar – The Kingston Trio

04

Evaluation

Metrics for Success

1. User satisfaction
2. Recommendation speed
3. Recommendation quality

05

DEMO (video)

https://youtu.be/H-LfMlVTpDU

06

Code Walkthrough

github.com/mariashapiro/soundalike2

Frontend + Backend

As mentioned earlier, we used React for our frontend and Flask for our backend.

This is app.js, which is the main file to tie front+back.

```
import './App.css';
import React, { useEffect, useState } from 'react';
import axios from 'axios'


import LandingPage from './pages/LandingPage';
import InputPage from './pages/InputPage';
import SmoothScroll from './components/SmoothScroll/SmoothScroll';


function App() {
  //const [getMessage, setGetMessage] = useState({})

  useEffect(()=>{
    axios.get('https://soundalike2.vercel.app/flask/hello').then(response => {
      console.log("SUCCESS", response)
      //setGetMessage(response)
    }).catch(error => {
      console.log(error)
    })

  }, [])
  return (
    <SmoothScroll>
      <LandingPage flexDirection="row"/>
      <InputPage flexDirection="row-reverse" />
    </SmoothScroll>
  );
}

export default App;
```

Server Code

We use Flask to handle business logic after the frontend makes a POST request.

```python
from flask import Flask, send_from_directory
from api.SearchSongHandler import SearchSongHandler
from flask_restful import Api, Resource, reqparse
from flask_cors import CORS, cross_origin #comment this on deployment
from api.HelloApiHandler import HelloApiHandler

app = Flask(__name__, static_url_path='', static_folder='frontend/build')
cors = CORS(app) #comment this on deployment
app.config['CORS_HEADERS'] = 'Content-Type'
api = Api(app)

@app.route("/", defaults={'path':''})
@cross_origin()
def serve(path):
    return send_from_directory(app.static_folder,'index.html')

api.add_resource(HelloApiHandler, '/flask/hello')
api.add_resource(SearchSongHandler, '/flask/search')
```

```python
class SearchSongHandler(Resource):
  def get(self):
    data = {
      'resultStatus': 'SUCCESS',
      'message': "Search Song Handler"
      }
    print('here')
    return jsonify(data)

  def post(self):
    SONG_TITLE_KEY = 'song_title'
    parser = reqparse.RequestParser()
    parser.add_argument(SONG_TITLE_KEY, type=str)
    args = parser.parse_args()
    print(args)

    song_title_value = args['song_title']
    # ret_status, ret_msg = ReturnData(request_type, request_json)
    # currently just returning the req straight
    status = "Success"

    if not song_title_value:
      status = "Unsuccessful"

    rec_songs, rec_song_metrics = get_recommended_songs_from_model(song_title_value)
    return jsonify({"status": status, "rec_songs": rec_songs, "rec_song_metrics": rec_song_metrics})

  def options(self):
    return build_cors_preflight_response()

def build_cors_preflight_response():
  response = make_response()
  response.headers.add("Access-Control-Allow-Origin", "*")
  response.headers.add('Access-Control-Allow-Headers', "*")
  response.headers.add('Access-Control-Allow-Methods', "*")
  return response
```

Server Code

We use Pickle to load and predict recommended songs using user input.

```python
def get_recommended_songs_from_model(song_title):
    PATH = os.path.abspath(os.path.join(os.path.dirname( __file__ ), '..', 'models/data'))
    unique_songs_file = pickle.load(open(PATH + "/unique_tracks.pkl", 'rb'))
    song_id_matches = unique_songs_file[unique_songs_file['song_title'] == song_title]['song_id'].tolist()

    rec_song_ids = []
    song_id = None
    if song_id_matches:
        song_id = song_id_matches[0]
    else:
        print("no song found. please try again")
        return rec_song_ids

    model = pickle.load(open(PATH + "/lmf_model.pkl", 'rb'))
    song_map = pickle.load(open(PATH + "/song_map.pkl", 'rb'))
    train_songs = pickle.load(open(PATH + "/train_songs.pkl", 'rb'))

    print("song id: ", song_id)
    rec_song_inds, rec_song_metrics = model.similar_items(song_map[song_id], N=6)
    print(rec_song_ids)

    rec_titles = []
    for rec_idx in rec_song_inds:
        song_id = train_songs[rec_idx]
        song_title = unique_songs_file[unique_songs_file['song_id'] == song_id]['song_title'].iloc[0]
        rec_titles.append(song_title)

    print('rec_songs: ', rec_titles)

    return rec_titles, rec_song_metrics.tolist()
```

Listening Matrix

Manipulate triplet listening data into item-user matrix where element (i, j) is number of times user j listened to i song.

```python
# construct sparse matrix of users and songs
def create_listen_matrix(load=True):
    if load and os.path.isfile('data/listen_matrix.pkl'):
        return pickle_load('data/listen_matrix.pkl')

    users, songs, listens = load_triplets('data/train_triplets.txt')

    # find unique users
    users_uniq = np.unique(users)
    # construct {user_id: index} dict for unique users
    users_map = {k: v for v, k in enumerate(users_uniq)}
    # map user_id to user_idx for every triplet user
    users_inds = list(map(lambda x: users_map[x], users))

    # find unique songs
    songs_uniq = np.unique(songs)
    # construct {song_id: index} dict for unique songs
    songs_map = {k: v for v, k in enumerate(songs_uniq)}
    # map song_id to song_idx for every triplet song
    songs_inds = list(map(lambda x: songs_map[x], songs))

    mat = sparse.coo_array((listens, (songs_inds, users_inds)), dtype=np.int32)

    pickle_dump(mat, 'listen_matrix.pkl')

    return mat
```

Model

Built wrapper model implementation from open-source library Implicit.

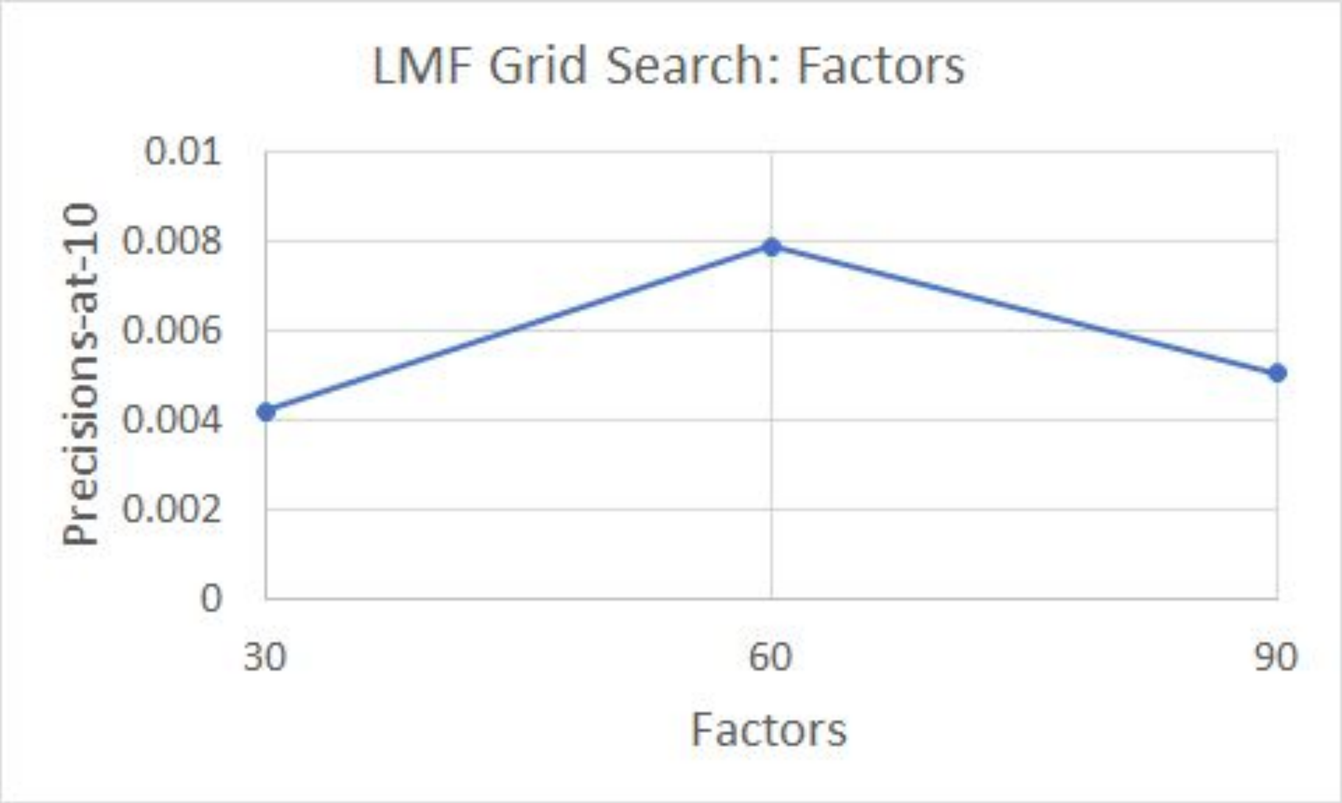
```python
    def __init__(self, type='cos', params={}, load=True):
        self.type = type
        self.load = load
        self.users, self.songs, self.listens = utils.load_triplets()
        self.song_titles = utils.load_song_titles('data/unique_tracks.txt')

        if type == 'cos':
            K = params.get('K', 10)
            self.model = implicit.nearest_neighbours.CosineRecommender(K=K)
        elif type == 'lmf':
            factors = params.get('factors', 30)
            lr = params.get('learning_rate', 1.00)
            reg = params.get('regularization', 0.6)
            iter = params.get('iterations', 30)
            neg_prop = params.get('neg_prop', 30)
            self.model = implicit.lmf.LogisticMatrixFactorization(
                    factors=factors,
                    learning_rate=lr,
                    regularization=reg,
                    iterations=iter,
                    neg_prop = neg_prop
                )
        elif type == 'als':
            factors = params.get('factors', 100)
            reg = params.get('regularization', 0.01)
            iter = params.get('iterations', 15)
            self.model = implicit.als.AlternatingLeastSquares(
                    factors=factors,
                    regularization=reg,
                    iterations=iter
                )
```

07

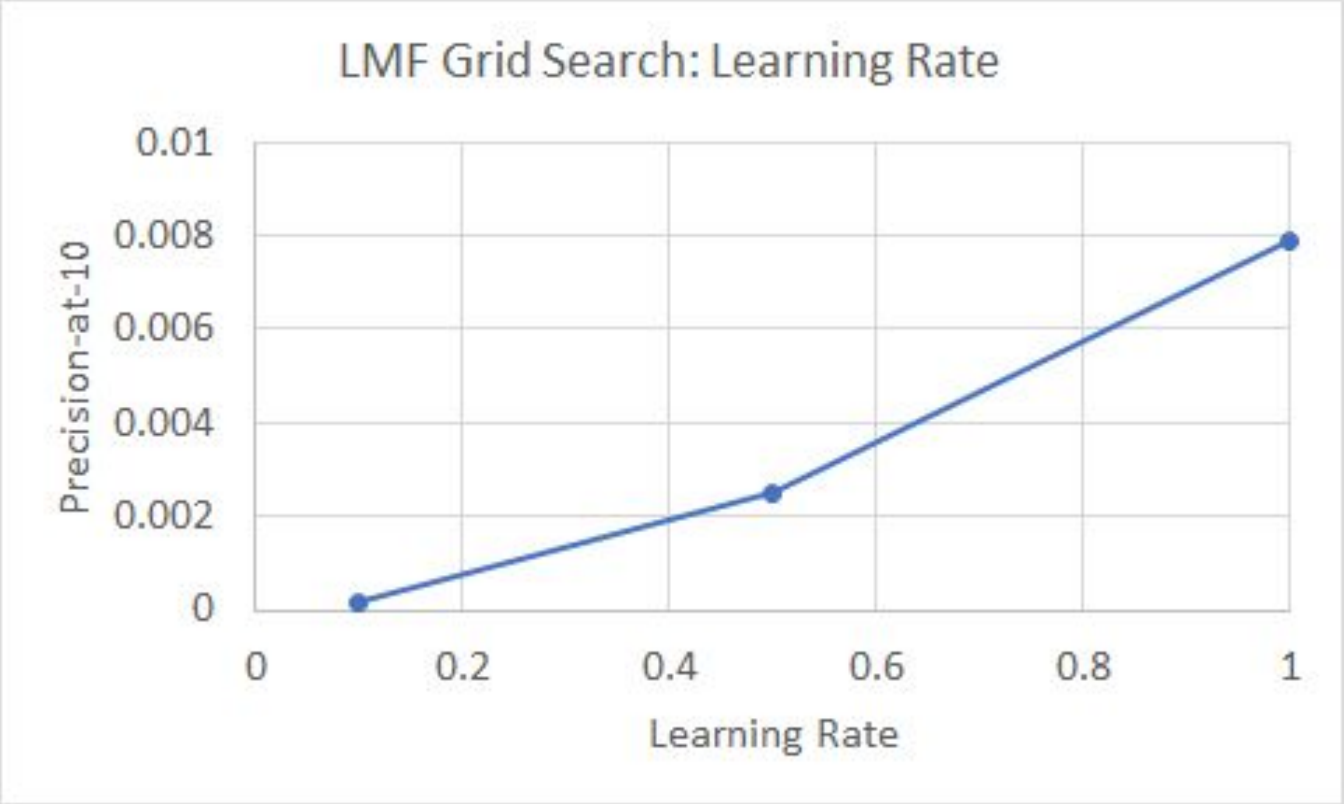
Evaluation Results

01 LMF Grid Search

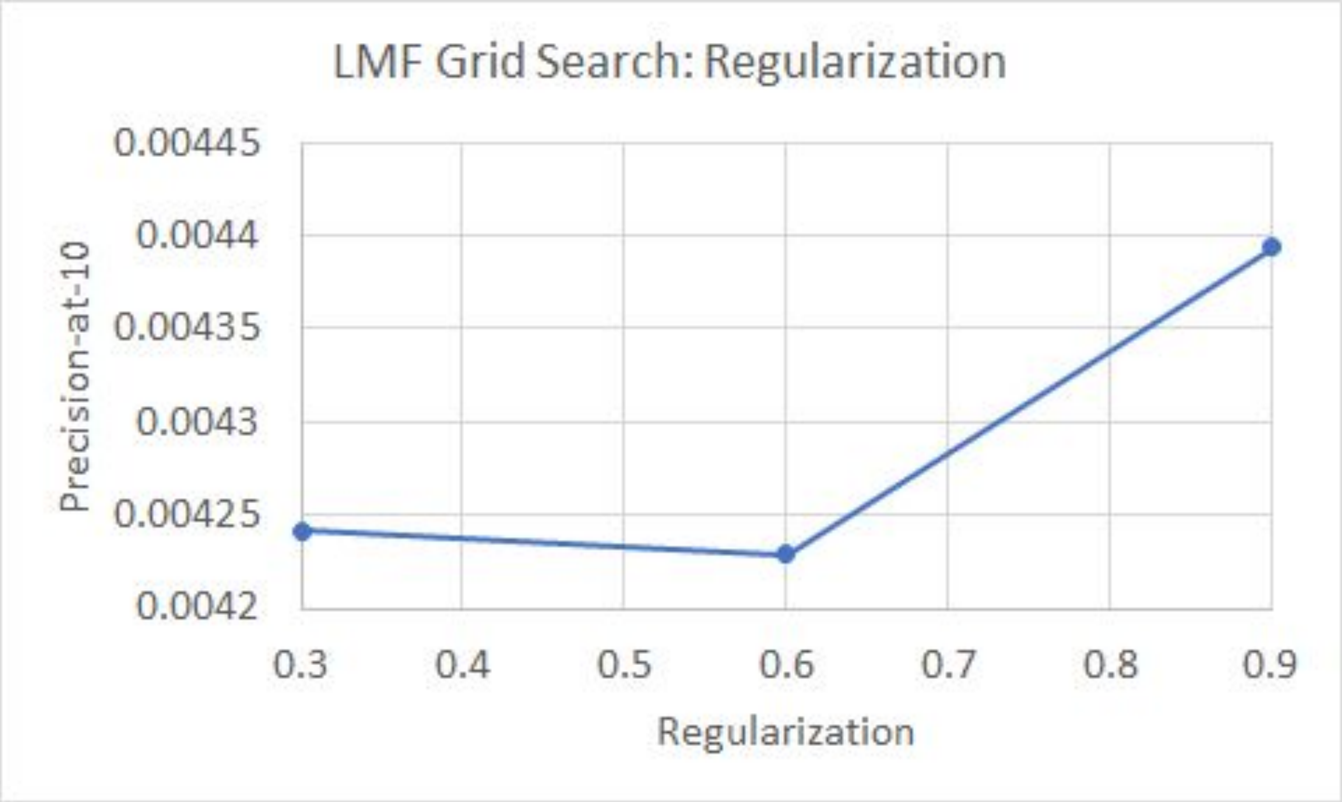
- Factors: [30, 60, 90]
- Learning rate: [0.1, 0.5, 1.0]
- Regularization: [0.3, 0.6, 0.9]



Learning rate = 1.0, Regularization = 0.6



Factors = 60, Regularization = 0.6



Factors = 60, Learning rate = 1.0

01 Selected LMF Model

Hyperparameters

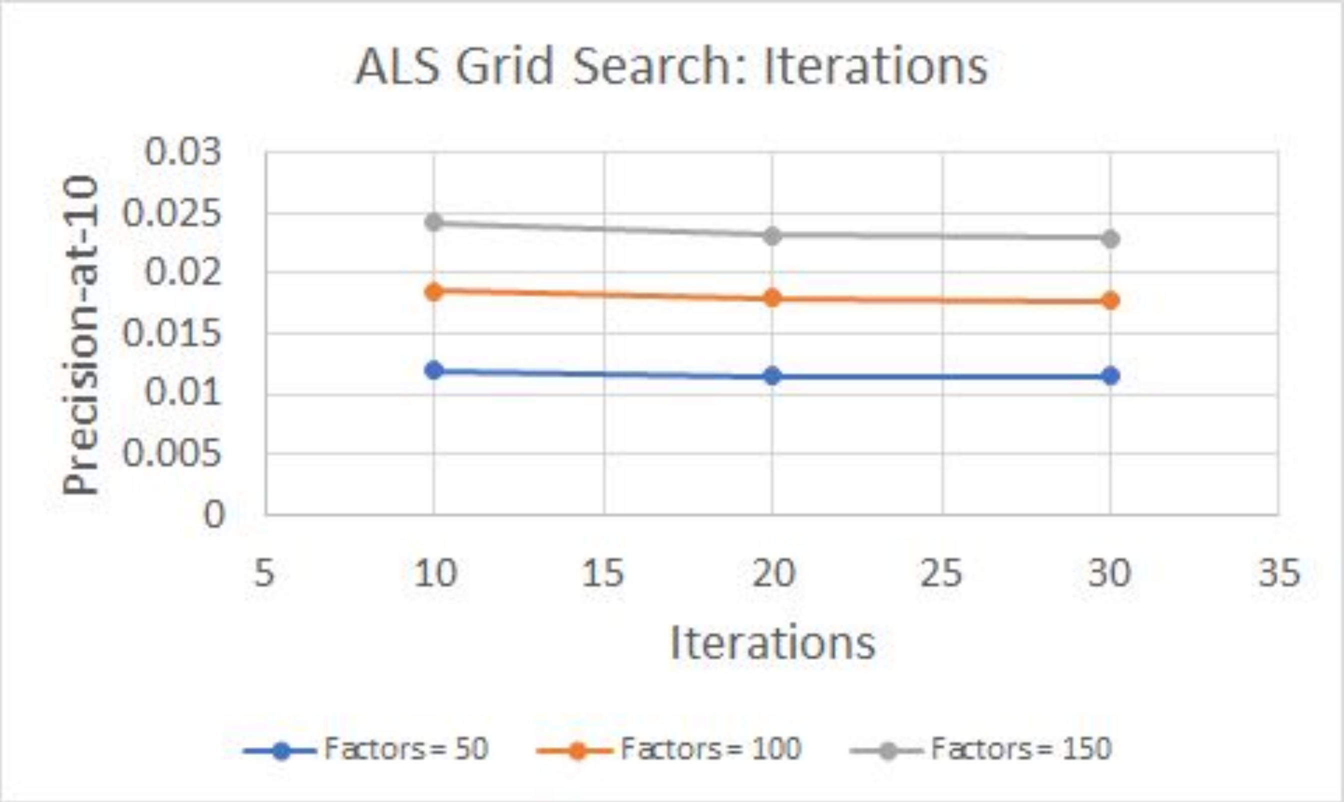
- Factors = 60
- Learning rate = 1.0
- Regularization = 0.9

Performance

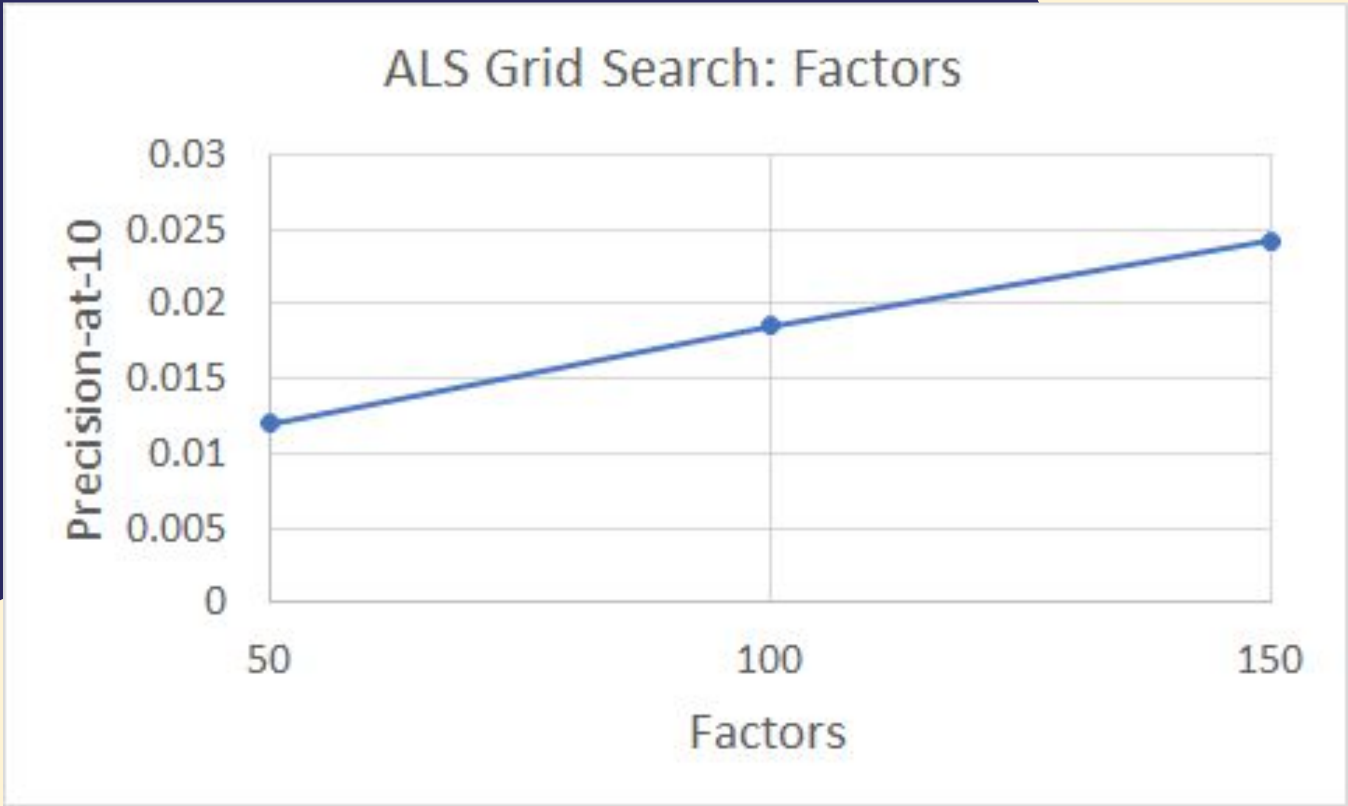
- Training time = 20 m 33 s
- Precision-at-10 = 0.8%

02 ALS Grid Search

- Factors: [50, 100, 150]
- Iterations: [10, 20, 30]
- Regularization: [0.001, 0.01, 0.1]



Regularization = 0.01



Learning rate = 1.0, Regularization = 0.01

02 ALS Selected Model

Hyperparameters

- Factors = 150
- Iterations = 10
- Regularization = 0.1

Performance

- Training time = 3 m 58 s
- Precision-at-10 = 2.4%

Deployed Performance

- Recommendation time: < 15 s
- Recommends five songs
  - Displays song titles
  - Also shows song similarity score

Problems Encountered

- Really big files -- the files associated with the dataset and the models were extremely large. We used the pickle library that serialized large models. These files were still too large and broke many things
    - Had to use github lfs to upload the files to github
    - These files broke our deployment, Vercel so we had to stick with a local deployment

- Handling virtual environments and ensuring the correct versions across our devices (referenced in our ‘open source’ section)
- Even though the million song dataset contains a million songs, the recommendation matrix still seems lacking

Best Features

- Fulfills our goal -- creating a recommender system using a large song dataset to create song recommendations
- Able to recommend a list of songs based using logistic matrix factorization
- Shows the similarity scores to indicate to the user how the song was selected

Future Improvements

- Adjust the payload from the frontend search bar to include artist as well for more accurate results
- Host the website on a server to utilize the large model files
- Incorporate the Spotify API

08

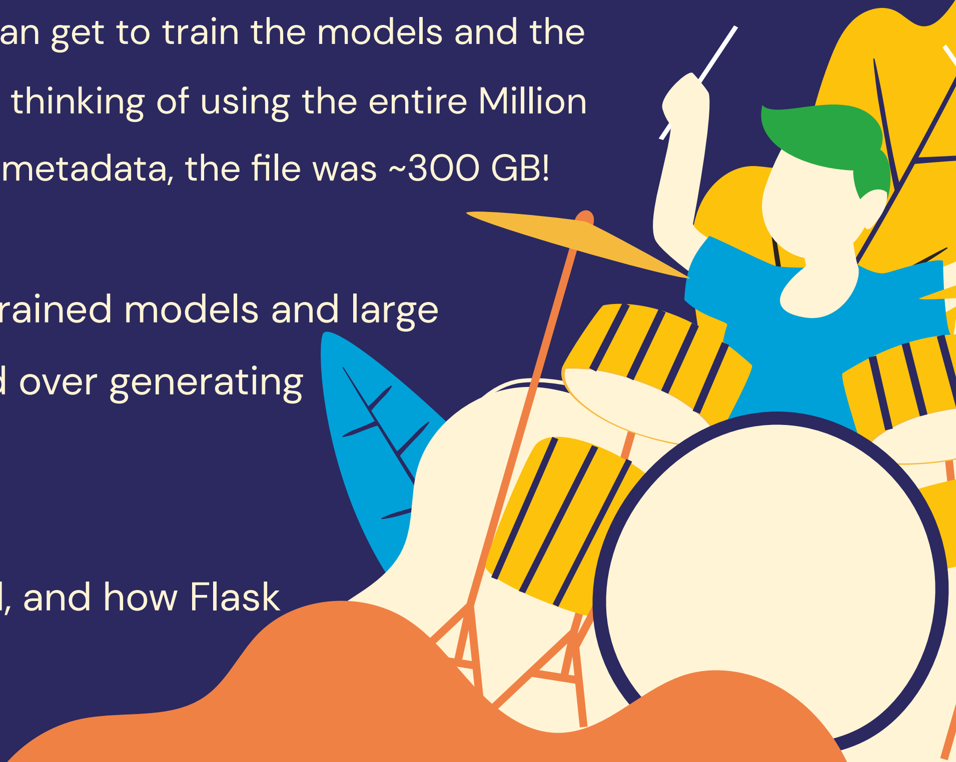
Lessons Learned

Lessons Learned

Maria: I didn't realize how big some files can get to train the models and the required storage resources. We we initially thinking of using the entire Million Song Dataset, and even though it just had metadata, the file was ~300 GB!

Robert: Using Pickle in Python to save trained models and large arrays/matrices greatly improves speed over generating them from scratch.

Ajay: I learned more about the backend, and how Flask React can complement each other.



Thanks!

Do you have any questions?

mas@gatech.edu
rmorgan61@gatech.edu
ajayvijayakumar2018@gmail.com

‘README’

Github link: https://github.com/mariashapiro/soundalike2

This includes all of our source code, including the Pickle file which is a condensed version of the models we used to generate recommendations. To run the website, please follow the instructions included in the repo Readme.

If you like our work, please cite us:

Maria Shapiro, Robert Morgan & Ajay Vijayakumar Spring 2022 Georgia Tech CS 4675 Final Project