Context-Aware, Scalable, Probabilistic Music Recommendation with Sampling and Clustering

Alexander Sfakianos alexander.sfakianos@case.edu
Department of Computer and Data Sciences
Case Western Reserve Univeristy
Cleveland, OH 44106-1712, USA

Daniel Shao daniel.shao@case.edu
Department of Computer and Data Sciences
Case Western Reserve Univeristy
Cleveland, OH 44106-1712, USA

Ryan Tatton ryan.tatton@case.edu
Department of Computer and Data Sciences
Case Western Reserve Univeristy
Cleveland, OH 44106-1712, USA

Abstract

Many modern streaming services seek to provide a continuous stream of content across a vast selection by utilizing recommendation systems. Recommendation systems seek to recommend new content based on various contexts, such as individual user preferences and platform-wide trends. We seek to create our own recommendation system, *connect.fm*, operating on top of the Spotify API to provide an alternative source for music recommendation. We utilize a unique component referred to as *stations* to establish connections between users based on their geographic location. Stations provide an alternative source of recommendations that works alongside more traditional recommender approaches. In addition to our stations, we interpret several contexts (user ratings, history, nearby users, and bias specified by the user) to group users. We generate a clustering of songs in our search space to facilitate an online learning environment. New recommendations are suggested through ancestral sampling in which our system samples over users, clusters, and songs.

Keywords: Recommendation system, Context awareness, Probabilistic graphical model, Clustering

1. Explanation and motivation

- What problem are you solving?
- What issue or topic are you investigating?
- Why is it interesting?
- What does success mean?
- How do we know if you are successful?

2. Introduction

The objective of a *recommendation system* is to provide "optimal" items to a user. We use the "ranking version" of a recommendation system described in (Aggarwal, 2016), which provides a comprehensive exploration of recommendation systems and their applications. The field of recommendation systems is diverse and rapidly developing with the accelerating usage of Internet and personalized services. Recommendation may be based on a variety of factors, including user attributes, item attributes, item-item relationships, user-item relationships, user-user relationships, and contextual data.

Recommendation systems are ubiquitous in music streaming services. We choose to develop a recommendation system based on Spotify's library of music and song features. With over 60 million songs, 280 million active users, and 144 million premium members, Spotify is the top music streaming service in the world. It offers rich features on both desktop and mobile platforms for browsing, creating, and sharing playlists. Consequently, we believe a recommendation system built from Spotify's enormous library and vibrant community will provide a novel and meaningful means to discover new music within the context of one's community.

connect.fm is an intelligent music streaming mobile application built with the Spotify API. A central concept in connect.fm is a *station*: a personalized music recommendation stream that utilizes the music preferences of the listener as well as those of nearby or invited listeners. By considering the music tastes of others, we provide Spotify users a new way to discover music and feel a closer connection with their community. This is a generalization over Spotify's current recommendation system, Discover Weekly, which provides a curated playlist based on an individual's listening history. Our system is distinct in that it considers the listening history of users and those in their geographical location, can incorporate user feedback to develop more precise understandings of user taste, and utilizes probabilistic clustering to recommend songs to users.

The recommendation system of connect.fm should be (1) probabilistic, to satisfy the nature of this course's project in addition to naturally addressing the uncertainty of recommendation; and (2) online, to allow for efficient and continual improvements to recommendations.

3. Related Works

In this section, we will provide an overview over existing clustering methods and recommendation systems, as well as commentary on each method's feasibility for our specific goal of generating song recommendations.

3.1 Recommendation systems

3.2 Clustering

Clustering is commonly applied in recommendation systems to create organization within a massive catalog of items. Consider Amazon's immense asssortment of products: each product is grouped into a category, such as cooking supplies, games, electronics, etc. Consumers who frequently shop for electronics will receive recommendations of items categorized as electronics. Evidently, the categorization of each item in a catalog is key to providing tailored recommendations while circumventing the need to exhaustively search the entire catalogue for suitable recommendations.

There exist an a variety of methods for unsupervised clustering outlined by Xu et al.

- *Hierarchical.* Dendrogram where the leaves are the data points and the interior nodes are cutoff criteria. This clustering is very intuitively understandable and data visualization is highly interpetable, but it fails for high dimensional data.

- *Partitional.* Define the number of clusters beforehand, and group objects into K clusters based on a similarity or distance metric. This method was also insufficiently efficient for high dimensional data.

- *Mixture model.* Assume each data point is generated from a unique probability distribution, where each probability distribution is a cluster. Different methods can be derived from different density functions (e.g., Gaussian vs t-distribution), or just have different parameters.

- *Graph-based.* Create a graph where each vertex is an item, and weighted edges represent the distance between two items. Numerous concepts in graph theory including mincuts and minimum spanning trees can then be applied to find suitable clusters. These methods are highly interpretable, but suffer from large reductions in running time as dimensionality increases.

- *Fuzzy.* Under this formulation, members can belong to multiple clusters. This provides an interesting insight, in which users with markedly different tastes may have some songs which they both enjoy. Under a fuzzy clustering approach, those shared songs can exist in multiple clusters with high likelihood.

We settle on the Dirichlet process mixture model (DPMM). These models are computationally efficient, which is essential to perform clustering of a large, high-dimensional dataset. Furthermore, the probabilistic nature of these models reflects the variance of user preference.

4. Methodology

4.1 Notation Convention

A non-bold, lowercase x denotes a scalar, function, or a generic variable. A non-bold, uppercase X denotes a random variable or constant. A bold, lowercase $\mathbf{x}$ denotes a vector or sequence of scalars. A bold, uppercase $\mathbf{X}$ denotes a matrix or column-wise sequence of non-scalars. A calligraphic, uppercase $\mathcal{X}$ denotes a set or distribution. We denote a single indexed valu as either $x_i \in \mathbf{x}$ or $\mathbf{x}_i \in \mathbf{X}$. For indexed sequences, we use the notation $x_{i:j} \in \mathbf{x}$ or $\mathbf{x}_{i:j} \in \mathbf{X}$ where the elements at indices i, j are included. Note that $x_{i:j}$ is a vector of dimension $(j-i) \times 1$ and $\mathbf{x}_{i:j}$ is a matrix of dimension $D \times (j-i)$ where D is the column dimension. Finally, a blackboard, uppercase $\mathbb{X}$ denotes a mathematical field.

4.2 Concepts

Before discussing the sampling and clustering components of our approach, we must define some terminology and concepts. A *song* $\mathbf{s} \in \mathcal{S}$ is a D-dimensional vector of song features, where $\mathcal{S}$ is the set of all songs that we can recommend. These features may be explicitly provided or implicitly learned. Additionally, they may be continuous, discrete, or some arbitrary combination thereof. We assume only continuous features. Namely, we use a subset of the available features provided by **?**:

$$\mathbf{s} = \begin{bmatrix} \textit{danceability} \\ \textit{energy} \\ \textit{loudness} \\ \textit{speechiness} \\ \textit{accousticness} \\ \textit{instrumentalness} \\ \textit{liveness} \\ \textit{valence} \\ \textit{tempo} \end{bmatrix} \in [0,1]^9 \tag{1}$$

All but loudness and tempo are already normalized between 0 and 1. For both loudness (-60 to 0 dB) and tempo (1 to 300[1] bpm), we apply min-max normalization,

$$\tilde{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

One of the aspects of a user is their music-listening *taste* $\mathbf{t}$, which is most generally defined as a function over their listening history,

$$\mathbf{t}_t = f(\mathbf{s}_{1:t-1}) \tag{2}$$

We use the simple yet, computationally-efficient approach of *exponential smoothing*,

$$\mathbf{t}_t = \alpha \mathbf{s}_t + (1 - \alpha)\mathbf{t}_{t-1} \tag{3}$$

where $\alpha \in [0, 1]$ is the smoothing factor. To incorporate the time at which a user listens to a song,

$$\alpha = 1 - e^{-\Delta t/\tau} \tag{4}$$

where τ is time-constant that we can use to vary the decay, and Δt is the elapsed duration between $\mathbf{t}_{t-1}$ and $\mathbf{s}_t$. Since Equation (3) is iterative, we can update the representation of in an online fashion and only maintain the single vector.

A user can give each song recommendation a *rating* $r \in \{1, 2, 3\}$, where 1 indicates the user listened to the song and rated it negatively; 2 indicates the user either listened to the song and did not rate it or has not listened to the song; and 3 indicates the user listened to the song and rated it positively. Additionally, we associate with each rating a *timestamp* $t \in \mathbb{N}_+$ to account its temporal relevancy, assuming that older ratings are less indicative of current user preference. We denote the timestamped rating history of a user as $\mathcal{R}$.

1. The choice of 300 bpm as the upper limit of tempo is arbitrary.

While our recommendation system draws upon music taste of other nearby users, we recognize that a user may want to control the degree to which we recommend based on the taste of other users. Perhaps the user *only* wants recommendations based on their taste at one moment, but suddenly decides to explore the local music preferences of those around them. Standard recommendation systems typically do not provide the user the ability to actively engage with the recommendation process. However, we believe that offering such control can improve user experience. We achieve this functionality in our recommendation system with *recommendation bias* $\beta \in [0, 1]$. When $\beta = 1$, we only recommend only based on the taste and ratings of the user. Conversely, $\beta = 0$ causes recommendations to only consider the taste and ratings of nearby users. If $0 < \beta < 1$, we linearly interpolate the preferences of the user and a nearby user.

Lastly, central to our approach is the *geolocation* g of a user, which may take the form of a latitude-longitude pair or a geohash. To summarize, we represent a user u as a 4-tuple,

$$u = (\mathbf{t}, \mathcal{R}, \beta, g) \tag{5}$$

With these concepts in mind, we can now discuss the details of our recommendation system.

4.3 Sampling

We call our user-specific, context-aware, online recommendation system a *station* to keep with the radio analogy. We conceptualize the recommendation process as sampling, where our objective is to efficiently sample from the joint distribution

$$p(\mathbf{s}, u, v, C, \beta) = p(\mathbf{s}|u, v, C, \beta)p(C|u, v, \beta)p(v|u)p(u) \tag{6}$$

where C is a cluster of songs and v is a nearby user or "neighbor." We discuss our approach to clustering songs in Section 4.4. We depict Equation (6) as a directed graphical model in Figure 4.3. To generate recommendations, we apply *ancestral sampling* (Bishop, 2006). That is, we first sample a neighbor $v \in \mathcal{N}$, then a cluster C, and finally a song $\mathbf{s} \in C$. In the degenerate case that a user u does not have any neighbors (i.e., $|\mathcal{N}| = 0$), then we simple let the user be their own neighbor, $u = v$.

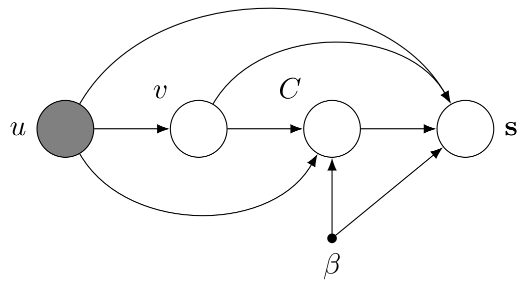

Figure 1: Directed graphical model of the connect.fm . We denote a (latent/observed) random variable with a (white/gray) circle and a deterministic parameter with a point.

Algorithm 1 defines our recommendation process. Because we represent user taste and songs as D-dimensional real-valued vectors, care should be taken when selecting an appropriate distance metric, particular as D becomes large (Houle et al., 2010). Notice that Equation (8) allows us to transform any symmetric, non-negative metric or distance function $d : X \times X \to \mathbb{R}_+$ into a normalized similarity metric $s : X \times X \to [0, 1]$ (Schubert, 2017). This is desirable, because we wish to associate *higher* probability with *similar* users, songs, and clusters. In normalizing, we sample based on *relative* similarity to a user u.

When sampling a cluster, and subsequently a song, we utilize an *adjusted rating* $\tilde{r} \in (0, 3]$ to integrate the concepts explained in Section 4.2. An adjusted rating is defined in Equation (11) and comprises of two forms of scaling: *similarity scaling* and *rating scaling*. We define the former in Equation (12), which accounts for the similarity between a song and the taste of the user. In other words, it indicates the overall "compatibility" of a song as a recommendation. Equation (13) defines the latter, integrates the ratings given by the user and their neighbor to specific song. As $|\mathcal{S}|$ grows, it is likely that most songs will either not be recommended to the user or not receive a rating an explicit rating. This is often the case in collaborative filtering techniques. Thus, this term will often evaluate to 2. For each rating, we apply *time scaling* using Equation (14). Assumption with such a weighting is that recommendation relevance is inversely proportional to the duration between the present and the time at which the user rated the song. The timestamp associated with the rating is the most recent occurrence of recommending the song. This is still the case if a user does not rate a recommendation since we assign all songs not given an explicit rating by the user a neutral rating of 2. Notice that since an adjusted rating cannot be 0, we avoid zero-probability clusters, thus alleviating the need to apply smoothing over the sampling distribution. Altogether, an adjusted rating allows us to encapsulate several factors that influence how probable a song is to be recommended.

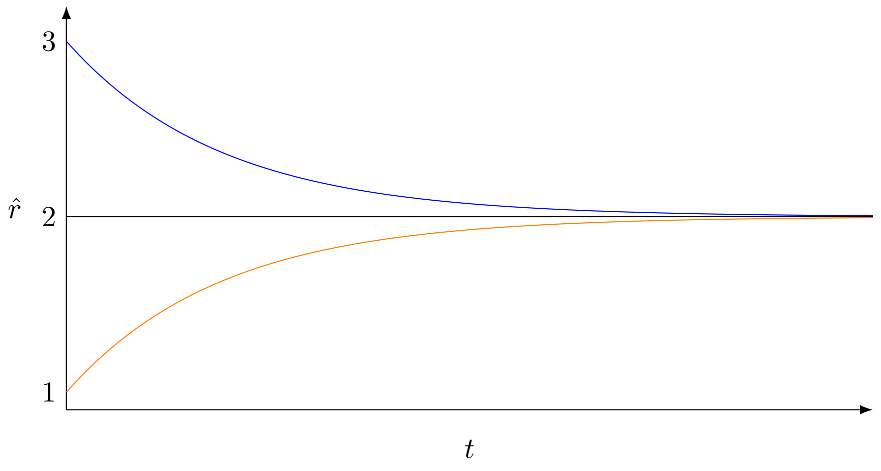

Figure 2: A conceptual view of how we combine rating and timestamp information to gradually decay the relevance of a rating. The upper and lower curves converge to the neutral rating of 2 over time, where the time-constant can be set to vary how rapidly we disregard the rating of a user.

Algorithm 1: connect.fm Recommendation System

1. Retrieve nearby users $\mathcal{N}$ within a distance R from user u such that

$$\mathcal{N} = \{v \mid d(g_u, g_v) \leq R, v \in \mathcal{U} \setminus \{u\}\} \tag{7}$$

where d is a suitable distance metric, and

$$s(x, y) = \frac{1}{1 + d(x, y)} \tag{8}$$

2. Sample a nearby user $v \in \mathcal{N}$ with probability

$$p(v|u) = \frac{s(\mathbf{t}_u, \mathbf{t}_v)}{\sum_{v \in \mathcal{N}} s(\mathbf{t}_u, \mathbf{t}_v)} \tag{9}$$

3. Sample a cluster $C \in \mathcal{C}$ with probability

$$p(C|u, v, \beta) = \frac{\sum_{\mathbf{s} \in C} \tilde{r}(u, v, \mathbf{s}, \beta)}{\sum_{C' \in \mathcal{C}} \sum_{\mathbf{s} \in C'} \tilde{r}(u, v, \mathbf{s}, \beta)} \tag{10}$$

where $\tilde{r}$ is the adjusted rating of song $\mathbf{s}$,

$$\tilde{r}(u, v, \mathbf{s}, \beta) = f(u, v, \mathbf{s}, \beta) h(u, v, \mathbf{s}, \beta) \tag{11}$$

$$f(u, v, \mathbf{s}, \beta) = \beta s(\mathbf{t}_u, \mathbf{s}) + (1 - \beta) s(\mathbf{t}_v, \mathbf{s}) \tag{12}$$

$$h(u, v, \mathbf{s}, \beta) = \beta \hat{r}(u, \mathbf{s}) + (1 - \beta) \hat{r}(v, \mathbf{s}) \tag{13}$$

$$\hat{r}(u, \mathbf{s}) = \text{sign}(r_{u,\mathbf{s}} - 2) \cdot e^{-\Delta t_{u,\mathbf{s}}} + 2 \tag{14}$$

and $\beta \in [0, 1]$ is recommendation bias.

4. Sample a recommended song $\mathbf{s} \in C$ with probability

$$p(\mathbf{s}|u, v, C, \beta) = \frac{\tilde{r}(u, v, \mathbf{s}, \beta)}{\sum_{\mathbf{s} \in C} \tilde{r}(u, v, \mathbf{s}, \beta)} \tag{15}$$

4.4 Clustering

One challenge for recommending songs is that the Spotify library contains 60 million songs. If we were to naively find a song recommendation, we would have to search the entire library for the best song, each time we wanted to recommend a song, which obviously is infeasible. To speed up the search, we chose to organize the songs into clusters, where similar songs are grouped into similar clusters. Using the assumption that if a user likes one song in a cluster, they'll like all songs in a cluster, we only need to find which cluster is most similar to a user's taste, and recommend songs based on that cluster. This vastly simplifies running time and storage complexity, and the nice thing is we only need to perform this clustering a single time, and we can use these clusters no matter how many users we have. For a single user, the running time becomes $O(K)$ rather than $O(N)$, where K is the number of clusters, and obviously K is vastly smaller than N.

When examining prospective clustering algorithms to employ for song clustering and user clustering, we considered the following requirements and desirable characteristics.

- *Non-parametric in the number of clusters.* Due to the immense size and high feature dimensionality of our clusters, it is difficult to qualitatively estimate the optimal number of clusters from the data. Consequently, we search for a method which implicitly optimizes the cluster count based on the data's structure. Although this optimization adds a constant cost to running time, its benefit in the quality of our clusters is well-worth the cost.

- *Computationally efficient in high dimensions.* Due to the immense number of entries and non-trivial dimensionality of our feature set, we require a clustering method which remains feasible at high dimensionality. This eliminates a large subset of clustering algorithms, including hierarchical clustering, k-means, and many density-based approaches.

- *Probabilistic.* We desire a probability density function-based approach in order to capture the variance in a user's music taste. A person's music taste varies each day with mood and environment, such that if we were to choose songs that perfectly match the user's music taste, then music would become repetitive. Furthermore, variety is a key component of the listening experience. If songs with the exact same cadence, rhythm, and beat were repeatedly recommended in a single listening session, the music would become predictable and boring. Consequently, non-probabilistic clustering methods such as k-means are undesirable, as the songs in a cluster would lack variance.

After a survey of potential clustering algorithms, we settled on the *Dirichlet process mixture model* (DPMM), one of the most popular probabilistic clustering methods for non-parametric clustering. This model fulfills the three requirements of being non-parametric, computationally efficient at high dimensions, and probabilistic. Intuitively, the Dirichlet mixture models function as a distribution of distributions. Under the DPMM, a Dirichlet probability distribution is defined for a set of probability measures, where each probability measure is itself a probability distribution over the sample space. In the context of our song space, each probability measure defines the likelihood that a song belongs to a given cluster, and the dirichlet distribution defines the likelihood of each probability measure to describe each song cluster.

We utilize a DPMM of Gaussian distributions with a symmetric Dirichlet prior. For a mixture of K Gaussian distributions and N samples, the model can be depicted as

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{i=1}^{N} \text{GMM}(\mathbf{x}_i) \tag{16}$$

where

$$\mathrm{GMM}(\mathbf{x}_i) = \sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j) \tag{17}$$

$$\begin{aligned}
\mathbf{x}_i &\sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}_i}, \boldsymbol{\sigma}_{\mathbf{z}_i}) \\
\mathbf{z}_i &\sim \mathrm{Cat}(\boldsymbol{\pi}) \text{ s.t. } \boldsymbol{\pi} = \{\pi_1, \cdots, \pi_K\} \\
\boldsymbol{\pi} &\sim \mathrm{Dir}(\boldsymbol{\alpha}) \text{ s.t. } \boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_K\} \\
\boldsymbol{\alpha} &\sim \text{Inv-Gamma}(\mathbf{1}, \mathbf{1}) \\
\boldsymbol{\mu}_j &\sim \mathcal{N}(\mathbf{0}, \mathbf{1}) \\
\boldsymbol{\sigma}_j &\sim \text{Inv-Gamma}(\mathbf{1}, \mathbf{1})
\end{aligned}$$

Just like in a Gaussian mixture model (GMM), we want to assign points to clusters according to the probability that each point belongs in a cluster. Formally, we assign x_i to the jth cluster through z_i which is the inferred index of a cluster. Although we initially specify a number of clusters, the final number of clusters will be inferred through optimization.

We use a multivariate Gaussian distribution as a mixture component and let $K = 60$. We optimize our model with *preconditioned stochastic gradient Langevin dynamics* (pSGLD), which is ideally suited for optimizing a large dataset through mini-batch gradient descent. pSGLD is defined as follows. Let $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j\}$ denote all model parameters. Then we can update the parameters of the ith iteration with a mini-batch size M,

$$\Delta\boldsymbol{\theta}^{(i)} \sim \frac{\epsilon^{(i)}}{2} \left\{ g(\boldsymbol{\theta}^{(i)})\mathcal{L}(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(i)}) + \sum_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(i)}) \right\} + \sqrt{g(\boldsymbol{\theta}^{(i)})}\mathcal{N}(\mathbf{0}, \epsilon_i \mathbf{1}) \tag{18}$$

$$\mathcal{L}(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(i)}) = \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}^{(i)}) + \frac{N}{M} \sum_{k=1}^{M} \nabla_{\boldsymbol{\theta}} \log \mathrm{GMM}(\mathbf{x}_k^{(i)}) \tag{19}$$

where ϵ_i is the learning rate for the ith iteration, $\log p(\boldsymbol{\theta}^{(i)})$ is the sum of log prior distributions over $\boldsymbol{\theta}$, and $g(\boldsymbol{\theta}^{(i)})$ is used to scale the importance of each iteration.

4.5 Complexity and Scalability

As a real-time music recommendation system, our approach must be consistently efficient with respect to both time and space. Moreover, its performance must be scalable to millions of songs and users. These requirements pose a significant challenge for any recommendation system, but are critical to address for the practical application.

4.5.1 Data Access

A key component of our implementation architecture is the Redis in-memory database (see Section 4.6.2), which provides both versatile data structures and unparalleled performance.

We provide complexity analysis specific to the sampling and clustering techniques in Sections 4.5.2 and 4.4.

Let N be the number of items involved in the execution of the command. To retrieve the values associated with keys

- `MGET`: $O(N)$
- `MSET`: $O(N)$
- `DEL`: $O(N)$ and $O(M)$
- `SADD`: $O(N)$
- `SRANDMEMBER`: $O(N)$
- `GEOPOS`: $O(N)$
- `GEOADD`: $O(\log N)$
- `GEOSEARCH`: $O(N + \log M)$ where N is the number of elements in the grid-aligned bounding box area around the shape provided as the filter and M is the number of items inside the shape.

4.5.2 Sampling

The number of neighbors we retrieve in Equation (7) is a function of the radius R, user location g_u, and total number of users $|\mathcal{U}|$. Intuitively, we will retrieve a larger number of neighbors if (1) the user u sets R to a larger value, (2) the user is located in a highly populated region, (3) the number of users increases, or any combination thereof. This affects the running time of Equation (9), which scales linearly in the number of neighbors, $\Theta(|\mathcal{N}|)$. Since we do not know *a priori* how many neighbors we will retrieve, it is prudent to limit the number of neighbors to $N_{\max} << |\mathcal{N}|$. With this modification, the running time becomes $O(N_{\max})$ since it is possible that only $N < N_{\max}$ are within a distance R from the user. In this way, we trade-off flexibility in recommendation for stability and time.

After sampling a neighbor, we then sample a cluster. Because we sum over all songs to compute the normalization term, Equation (10) is the most computationally expensive step in the recommendation process and has a running time of $\Theta(|\mathcal{S}|)$. This complexity becomes prohibitive, even for a modest number of songs. To resolve this problem, we propose sampling at most $K_{\max} << |\mathcal{S}|$ from all clusters such that we sample

$$K_i = \max\left\{|C_i|, \left\lceil \frac{K_{\max}}{|\mathcal{C}|} \right\rceil\right\} \tag{20}$$

from the ith cluster. We can show the validity of this approach by letting $X, \widehat{X}$ be the true and approximate values of the numerator in Equation (10), respectively. Moreover, let $Z, \widehat{Z}$ be the true and approximate normalization terms, respectively. Then, the percent error ε of an approximate cluster score is defined as

$$\varepsilon = \frac{Z}{X}\left|\frac{X}{Z} - \frac{\widehat{X}}{\widehat{Z}}\right| = \frac{Z\left|\widehat{X}Z - X\widehat{Z}\right|}{XZ\widehat{Z}} = \frac{\left|\widehat{X}Z - X\widehat{Z}\right|}{X\widehat{Z}} \tag{21}$$

First, note that we may sample all $|C_i|$ songs and still have non-zero error since $Z, \widehat{Z}$ depend on all clusters. Second, observe that $\widehat{X}, \widehat{Z}$ are relatively fixed for a given $K_{\max}$, depending on the variance of the song feature vectors in the cluster. Thus, as $|\mathcal{S}|$ increases, X, Z will dominate. Let us assume that each cluster is equiprobable, $X = \frac{Z}{|\mathcal{C}|}$. Then

$$\varepsilon = \frac{|\mathcal{C}|}{Z\widehat{Z}} \left| \widehat{X}Z - \frac{Z\widehat{Z}}{|\mathcal{C}|} \right| = \left| \frac{|\mathcal{C}|\widehat{X}}{\widehat{Z}} - 1 \right| \tag{22}$$

Thus, unless the true probability deviates significantly from the uniform distribution, $X \propto p(C|u, v, \beta)$ will be reasonably approximated. Practically, the cluster probabilities will not be equiprobable and of equal size, so varying degrees of under- and over-estimation will occur.

Once we sample a cluster, the last step in our recommendation process is to sample a song. We again sample K_i songs in accordance with Equation (20) and use the result from sampling a cluster to validate the approximate probabilities of the sampled songs. It is evident that the running time of this step is $\Theta(K_i)$. While limiting the number of sampled songs to K_i, we repeat this process for each recommendation, and so each recommendation will be based a different K_i-subset of songs. Further, so long as their is sufficient storage to store user data, song features, and cluster assignments, the time complexity of our sampling approach is *fixed*, regardless of $|\mathcal{S}|$ or $|\mathcal{U}|$.

4.5.3 Clustering

While a simple, brute-force method for song recommendation would be to exhaustively search the song space $\mathcal{S}$ for songs with characteristics matching the user's taste, although this method would run in O(n) time for a single user, it is still computationally expensive due to Spotify's enormous music library of over 60 million songs. For the problem of k users requesting song recommendations, the O(kn) problem quickly scales to become intractable for even a small increase in users requesting songs O(kn). Evidently, the processing of 60 million songs is the bottleneck. One option to hasten the song recommendations for multiple users would be to sort the music library into clusters, such that each cluster consists of songs with similar characteristics. Given these clusters, we can simply identify which cluster is most likely to be enjoyed given a user's music tastes, and recommend them songs from that cluster. Consequentially, we would only need to cluster the music library once, then out of c total clusters, identify the most fitting clusters for each of k users. After clustering, this would take O(ck) time, where c ¡¡¡ n. Let f(n) be the time required to complete clustering. Then the final running time would be O(f(n) + ck), which will be an improvement over the original O(kn) provided that f(n) ¡ kn. Notably, the running time for Dirichlet mixture modeling is based on the number of max iterations parameter, so one can simply set f(n) to be less than kn to achieve an improvement in running time.

4.6 Implementation

As part of the connect.fm mobile application, the source code for this project can be found at `https://github.com/connectfm/cfm/tree/main/backend`.

4.6.1 Architecture

4.6.2 Data Storage

From the official documentation, “Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams.”

4.6.3 Sampling

4.6.4 Clustering

Dirichlet Mixture Model clustering was implemented with Tensorflow version 1.13.1 and Tensorflow-probability version 0.6. Stochastic Gradient Langevin Dynamics was implemented with Tensorflow-probability’s optimizer library.

5. Results

- What are the take home messages from your project?
- Did your approach “work”? Why or why not?
- How could it be improved?
- What did you learn from the project?
- What other approaches might you consider if you did further work on it? There were numerous adaptable decisions made for sampling, learning, and clustering. These choices and alternatives are further described in the following section: Future Directions.

6. Future Directions

- Learning component
- Metric learning
- Handling of self-biased recommendations - if a user only uses connect.fm; this affects how we compute taste.
- “Closed” session with invited neighbors only.
- Experimenting with different Redis schemas to optimize time-space complexity tradeoff.
- Clustering with different methods

We are lastly interested in employing different clustering methods. One promising alternate clustering algorithm which suits our data is ρ-approximate Density-Based Spatial Clustering of Applications with Noise (ρ DBSCAN), which applies density based clustering, requires only one hyperparameter with clear heuristics for deciding an optimal value, and has an average running time of $O(n \log n)$. DBSCAN satisfies our requirements, and also contains an added benefit of identify clusters of arbitrary shape. In contrast, traditional clustering algorithms typically search for spherical clusters. Furthermore, DBSCAN is built to consider the notion of noise, which aligns well with the inconsistent nature of music taste. Two songs with identical song characteristics may evoke completely different reactions from users due to factors such as variance in the mood and environment of the user.

First, we will provide an intuitive explanation of DBSCAN. In essence, the DBSCAN algorithm defines clusters as regions of high density items, with sparse regions separating each cluster. To define these high density regions, the following definition criterium must be met: For each point in a cluster, there must exist sufficient items within a certain radius from that point.

To understand the mathematical basis behind how DBSCAN identifies clusters, we present five key definitions

Definition 1 (Core point) *A point p is a core point with respect to MinPoints and ϵ if there are $\geq N$ points within ϵ distance of p.*

Definition 2 (Directly density-reachable) *A point p is density-reachable from a point q if p is within epsilon distance from q, and q is a core point. distance.*

Definition 3 (Density-reachable) *A point p is density reachable from q if there is a path from p to q, in which each point in the path is directly-densit -reachable to its neighbors in the path.*

Definition 4 (Density-connected) *A point p is density-connected to point q if there is a point o which is density reachable to p, and density-reachable to q.*

Using these definitions, the original DBSCAN algorithm defined a cluster as a set of density-connected points which is maximally density-reachable. Notably, this definition of clustering results in the exlcusion of points which are not density-connected to enough points with respect to epsilon and N. These sparsely connected points will not be part of any cluster, and are treated as noise. We adapt this definition slightly to fit our needs better, as we seek to make use of all songs. Consequently, we let each noisy point be its own cluster. Furthermore, we utilize an approximate DBSCAN algorithm which makes a small sacrifice in accuracy in order to vastly expedite running time, from $\Omega(n^{4/3})$ to $O(n)$ time.

Definition 5 (ρ-approximate density-reachable) *A point q is ρ-approximate density-reachable from p if there is a path from p to q such that all points other than q are core points, and consecutive vertices along the path have distance less than $\epsilon(1 + \rho)$*

Definition 6 (ρ-approximate cluster) *A ρ approximate cluster C is a nonempty set of points which satisfy two requirement.*

1. Maximality: If a core point p is in C, then all points density-reachable from p are also in C

2. ρ-approximate Connectivity: For any two points, p, q in C, there exists a third point z such that p and q are both ρ-approximate density-reachable from z.

7. Conclusion

8. Individual Contributions

For Daniel and Ryan, this project also relates to their senior capstone on implementing the mobile application, connect.fm. While Alex is not officially involved in the capstone project, he has helped implement some of the AWS architecture of the recommendation system backend. Specifically for this project, the following is a description of individual contributions from each member. All members contributed equally toward reviewing the research literature.

8.1 Alexander Sfakianos

- Implemented the evaluation code of our recommendation system under the `evaluation` package on the connect.fm GitHub repository.
- Implemented the processing of incoming data from connect.fm users under the `dbprocessing` package on the connect.fm GitHub repository.

8.2 Daniel Shao

- Implemented the clustering model under the `clustering` package on the connect.fm GitHub repository.

8.3 Ryan Tatton

- Unofficial team lead for the project, helping ensure its succesful completion.
- Formalized the recommendation system algorithm and data representation.
- Implemented the recommendation system provided under the `recommendation` package on the connect.fm GitHub repository.

References

Charu C. Aggarwal. *Recommender Systems: The Textbook.* Springer International Publishing, 1 edition, 2016. ISBN 978-3-319-29657-9. doi: 10.1007/978-3-319-29659-3. URL `https://www.springer.com/gp/book/9783319296579`.

Christopher M. Bishop. *Pattern Recoginiton and Machine Learning.* Springer International Publishing, 1 edition, 2006. ISBN 978-0-387-31073-2.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231, Portland, Oregon, August 1996. AAAI Press.

M. Houle, H. Kriegel, P. Kröger, E. Schubert, and A Zimek. Can shared-neighbor distances defeat the curse of dimensionality? In *Scientific and Statistical Database Management,* *SSDBM 2010*, volume 6187, pages 482–500, 2010. ISBN 3642138179. doi: 10.1007/978-3-642-13818-8_34.

Erich Schubert. Knowledge discovery in databases [PDF document], 2017. URL `https://dbs.ifi.uni-heidelberg.de/files/Team/eschubert/lectures/KDDClusterAnalysis17-screen.pdf`.