MUSICAL FEATURE SPACE: USING INFERENCE TO EVALUATE AND RECOMMEND SONGS BASED ON USER PREFERENCES

THOMAS LEECH AND WILLIAM LOUCKS

Abstract. This paper aims to identify an effective means for predicting an individual's musical preferences. First, we examine an inference method to predict how a user will score a given song, based on historical preferences, and then we present a sampling approach to recommend songs that the user is likely to enjoy. Using a dataset that parameterizes musical attributes from a million songs, we employ Markov Chain Monte Carlo sampling to evaluate a user's affection for carefully selected features. With those features, we illustrate an effective way to predict a user's affinity toward a particular song, and as a consequence, a means to identify songs that align with a user's demonstrated preferences.

Contents:

1. Prior Work

Assessing the most effective way to recommend material to users based on demonstrated preferences is a challenge of keen interest within the consumer product industry. Whether its food, clothing, movies, or songs, showcasing material that a user is likely to consume helps retain interest. For example, in late 1999, Pandora Media created the Music Genome Project to isolate 450 musical attributes within a given song for the purpose of using the information to queue a unique list of song recommendations for each user. Since then, Pandora has patented the technology, registered the name as a trademark, and refused to release its list of 450 attributes, making the project's parameters a trade secret.[1] Pandora's ability to appropriately suggest music to its customers is crucial to its business model, and revealing the attributes would likely compromise the uniqueness of the approach. However, while the magic of Pandora's music recommendation system rests in the ability to extract and manipulate large amounts of musical information, other companies have taken different approaches.

In 2007, Songza introduced a digital, on demand music recommendation platform that uses a curated crowd of individuals to create playlists with certain themes. The packets of songs used to be created by trusted experts who shape the playlists to fit particular events, moods, and contexts. Moreover, while Songza hasn't been directly available since late 2015, its services have since been folded into Google Play Music, likely indicating a continued interest in the approach.[2][3] Notably, and in contrast to Pandora, Songza's approach relies on a human to decide which songs are most similar, and by extension, which songs a user might enjoy given that user's temporal preferences. Although, while Songza and Pandora employed successful audio recommendation platforms, many services don't just stick to one technique.

Using a multifaceted approach, Spotify, the music streaming service with over 100 million users, employs three techniques to recommend songs to its users: analysis of musical features, web searching, and collaborative filtering. Spotify takes Pandora's approach of decomposing songs into relevant features, such as tempo and loudness, to assess how likely a user is to enjoy a given song. However, Spotify only uses this technique for new songs that haven't had enough time to develop a base of listeners. Moreover, the audio streaming company also uses natural language processing platforms to explore websites for metadata, incorporating words that are associated with particular songs and artists into its models. Lastly, collaborative filtering allows Spotify to crowd source assistance for its recommendation algorithm. In collaborative filtering, a user's listening history will be compared to the preferences of other users who have a similar listening history.[4] The concept of collaborative filtering is illustrated in Figure 1.

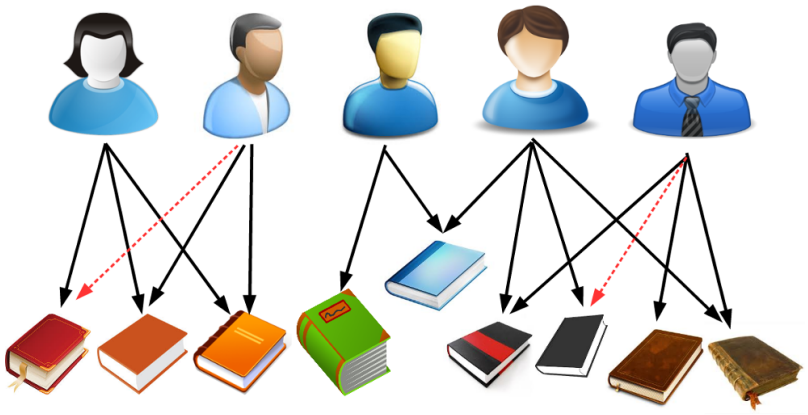

FIGURE 1. **Collaborative Filtering:** The black arrows represent known preferences for each consumer. The red arrows indicate suggestions made by the producer, based on the preferences of consumers with similar interests.[5]

According to Spotify, Discover Weekly, the playlist of recommended songs that Spotify genereates for every user each week, was used to stream over 5 billion songs from July 2015 to May 2016.[6] As a result, it's hard to doubt the success of the threefold method to suggest songs that the audio streaming service employs. However, there is a human element, allowing for user feedback, that Spotify lacks on most of its services (its Radio service allows users to give feedback). Netflix, for instance, also uses collaborative filtering (to recommend movies and TV shows); however, the video streaming service offers a "thumbs up", "thumbs down" rating feature, which adds a *user* factor to the recommendation algorithm.[7] Netflix augments the collaborative filtering process, likely to hone in on users who have not only streamed the same video as the user of interest, but also rated movies and shows in a similar way. The benefit of this system is to prevent the algorithm from believing that a user's history represents his or her preferences, since a user could have streamed unsatisfactory media. On the other hand, a user may not rate a show or movie every time after watching, decreasing the effect of the feature. However, the added opportunity to score media increases the recommendation algorithm's confidence in its suggestions and possibly allows the model to deliver more optimal suggestions.

2. Introduction

As mentioned, the premier music streaming services largely do not incorporate user feedback in their models for suggesting songs, as Netflix does in its movie and TV show recommendation algorithm. We would like to extend the prior work in suggesting songs by modeling user's preferences for particular song features, like Pandora and Spotify, and by incorporating Netflix's approach of direct user feedback. We recognize the success of Spotify and Pandora, and we would like to determine if user feedback, in the form of a rating from 1-10 on a particular song, enhances song recommendation capability for short temporal periods. In the end, our model will yield a novel way

to predict an individual's scoring of a song and a technique to recommend songs to users. Moreover, our method will shed light on which musical properties may be the most indicative of user preference.

We recognize that a user's immediate preference for a particular type of song may be dependent on contextual factors outside of a his or her historical preference, such as mood, song theme, time of the year, and time of day. However, collaborative filtering and the other techniques employed to suggest songs largely ignore the aforementioned contextual information that surrounds a user's temporal preferences. Our method, recommending songs to a user based on preferences within a short temporal period, allows for consideration of the aforementioned short-term factors. However, exclusively recommending songs based on recent preferences presents difficulty when accounting for long-term historical preferences.

3. The Model

Our approach, in both cases, rests in our model's ability to assess a user's musical preferences, based on a user scoring system. Moreover, each time a subject uses our model, the model resets to allow for variation in musical preference. For instance, if a user wants to listen to slower music at one particular time, our model would appropriately account for the temporal preference. On the other hand, if the same user where to employ the model later in the day, slower music would not be immediately accounted for, allowing for variation in context.

Our model examines 5 musical features, thrusting the features into what we call feature space. Feature space is a five dimensional space where each dimension is represented by one of the musical features. The five musical features, which we also call attributes, considered are timbre, song *hotttnesss*, loudness, tempo, and year of production. While there may be other factors that could influence a user's affection for a particular song, the million song data set that we used, provided by Columbia University and created by MIT Media Lab spin-off company, The Echo Nest, most appropriately characterizes these five features for our purpose.[8] We believe that these five musical attributes can have second and third order effects when indicating user preferences, yielding more information about a user than just his or her penchant for the five attributes. The definitions of the attributes, and their possible higher order effects, are as follows:

Timbre: *A metric used to distinguish different sonic qualities. This metric could indicate the presence, and extent, of particular instruments, such as piano, vocals, guitar, and others. Furthermore, the presence of certain instruments produces implications about about genre and musical theme.*

Song *Hotttnesss*: *The Echo Nest generates this metric based on a song's popularity. Hotttnesss is used to indicate whether or not a user has preferences for songs and artists that are more well known. Additionally, hotttnesss groups artists into tiers, and a user might find his or her preferences in a particular band of popularity.*

Loudness: *The sound intensity of the song. This metric could indicate if a user likes soft songs, or louder songs, which can be suited for different contexts or moods. Loudness can also indicate which themes and tones a user prefers.*

Tempo: *The number of beats per minute in the song. Beats per minute indicates the speed of the song, which might vary depending on the context of listening– e.g. with friends or alone– and the mood of the listener.*

Year of Production: *The year the song was made publicly available. A user may have a preference for a particular era of music, some eras having distinct tones which contains attributes beyond the ones in our feature space.*

To gather a user's preferences, we randomly select 10 songs from our database and ask the user to rate the 10 songs on a scale from 0-10. A scoring of 0 indicates that the user does not like the song and would not like to listen again. Moving up, a 5 indicates that the user thinks the song is average and holds largely indifferent sentiments towards it, while a 10 indicates that the user

enjoys all aspects of the song. Any scoring in between indicates varying levels of satisfaction and enjoyment the user gets while listening. This description was provided to each participant. After gathering the user's scoring for each of the 10 songs, we add the information to our model and update our multivariate distribution, based on the features within that song. Ultimately, the user simply provides one numerical rating, from 0-10, for the entire song, and our model updates each feature accordingly.

One assumption is the constancy of the user's rating across all features. For example, if a user assigns a song a score of an 8, our model assumes that the user rates all five of the attributes at the level of an 8. We assess that a user's affinity to particular features within the song may vary, but our selected features are broad in scope, and are unlikely to vary too much beyond the overall score. For instance, a user could score a song at an 8 because the vocals were excellent, despite the guitar deserving a score of 6; however, our model does not isolate guitar from vocals, instead treating them together in timbre. While this may present an issue, we assess that this also allows the model to avoid assuming too much about a single aspect of a song, avoiding over-fitting to a single feature and instead examining the song holistically. In addition, our model accounts for the covariance between particular features, giving the model an understanding of how to adjust in a particular dimension, given that the features are not independent.

Moving on to the construction of our model, our multivariate distribution lies in the aforementioned five dimensional feature space. Our prior distribution, before a user scores the first song, is Gaussian over each of the five features, centered at the average value from the million song dataset. Moreover, our covariance matrix represents the covariance between each of the five features in feature space, among the songs in the dataset. We determined that a Gaussian model, updated using a Markov Chain Monte Carlo approach, is the most appropriate model for our context. Each of the individual features is roughly Gaussian distributed over all possible values in one dimensional feature space. As mentioned, we combined the five selected features for a particular song into a five dimensional vector and established a multivariate Gaussian distribution to represent a user's preferences for the song features.

The million song dataset was structured in a way that complicated the calculation of an overall average value for each feature, so we randomly selected a sample of 1000 songs to estimate the average feature vector. The covariance matrix was determined using the same subset of 1000 randomly selected songs. In addition, we specified a drift kernel as a Gaussian distribution over the previously sampled values. We found that the posterior distribution sometimes placed little significance on some areas strongly represented in the prior, and we use the drift kernel to hone in on areas with higher probability, accounting for the user's preferences. From sample to sample, our model estimates the score a user is expected to assign to a particular song, calculates the difference between the expected score and the user's actual score, and adjusts the multivariate distribution accordingly.

To measure the distance between scorings, we first examine the distance in feature space between a scored song and a sampled song from the multivariate distribution, through the following:

$$\vec{\Delta} = \frac{\vec{p} - \vec{f}}{\vec{\sigma}}$$

Where $\vec{p}$ is a five dimensional vector sampled from the multivariate Gaussian in our feature space, representing the distribution of the users preferences. $\vec{f}$ is the five dimensional feature vector for a song in the 1000 song subset that a user has scored, and $\vec{\sigma}$ is a five dimensional vector where each entry is the standard deviation for each respective feature (operations are performed piece-wise for each feature). This operation equalizes differences in traveling within each dimension of feature space, and the division allows us to use a dot product, or cosine distance formulation. Now, to associate the above distance with an expected rating for $\vec{f}$, we use the following:

$$\text{Expected Score} = 10e^{-c\vec{\Delta}\cdot\vec{\Delta}}$$

Therefore, as the squared difference between a sample of the user's preferred feature values and the song's feature values decreases, the expected score gets closer to 10. Conversely, as the difference grows, the expected score approaches 0. The constant, c, in the exponent is employed to

cater our function to this particular context. Through testing, we believe $c = 0.1$ most appropriately characterizes our model.

In the end, to update the multivariate Gaussian in feature space, we create a likelihood function that is a Gaussian distribution, with the Expected Score as its mean and 0.001 as its standard deviation. The small variance prioritizes songs with a small distance to the user's preferences, a value we honed through testing. We use observe statements in the Webppl language to specify the user's assigned score, and through Markov Chain Monte Carlo (MCMC) sampling, we infer a new multivariate distribution that shifts in feature space to minimize the difference between Expected Score and the actual given score in the observe statement. Moreover, the aforementioned drift kernel allows our model to sample more frequently from probabilistic areas with more mass, avoiding Markovian traversal of a chain with low probability paths.

Using MCMC inference, we burned 10,000 samples to avoid obtaining our posterior too early in the inference, increasing the likelihood that we converge on the appropriate values. In addition, we used a lag of 10 to further our chances of optimal convergence. An example of the two-dimensional relationship between features is shown in Figure 2. In this example, we used 1000 samples with a lag of 10 but a burn of 0 to show the path to convergence. The figure depicts the relationship between Tempo (on the x-axis) and Timbre (on the y-axis). Tempo is defined in beats per minute (BPM) and Timbre is represented in arbitrary units based on the information collected by the Echo Nest. The graph nicely shows an example of the inference converging on a posterior distribution with a Tempo of about 145 BPM and a Timbre around 5.9. Note that the circles far from the center value, the value that the inference converged on, are lighter, meaning they were sampled less frequently. Moreover, as the points get closer to the convergence, the circles get darker, indicating more sampled points near that value.

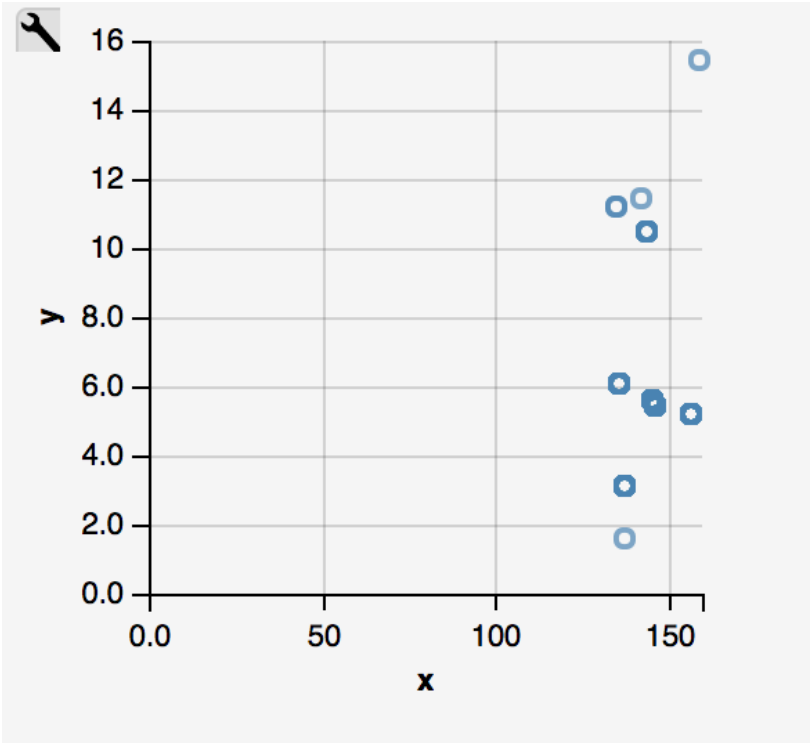

FIGURE 2. **Two-Dimensional Posterior** (Samples: 1000, Burn: 0, Lag: 10): MCMC inference displaying a posterior comparing Tempo (x-axis) to Timbre (y-axis) and converging on a data point.

In Figure 3, we used a burn of 10000, allowing our model to converge onto an area more quickly with 1000 samples. Typically, we observed only 1 to 3 dark points on the plots generated using a

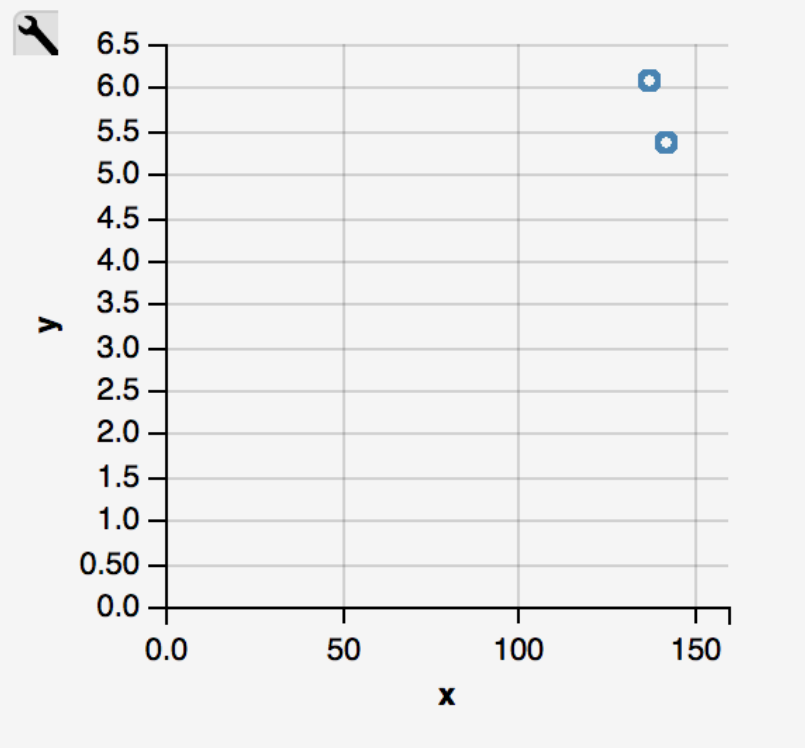

FIGURE 3. **Two-Dimensional Posterio**: (Samples: 1000, Burn: 10000, Lag: 10): MCMC inference displaying a posterior comparing Tempo (x-axis) to Timbre (y-axis) and converging on a data point.

burn of 10000 to estimate the posterior of a user's preference in two dimensions. In application, this pattern of inference was used to validate our model by predicting a user's scoring of a particular song. Furthermore, we used the inference model to recommend songs by measuring the feature distance between a sampled feature vector from user's multivariate distribution and the feature vectors of other songs in our dataset.

3.1. Predicting User Scoring. To predict how a user will score songs in the future, we ask the user to score 15 songs. We use the first 10 songs to train our model, and then we test using the last 5 songs. Our multivariate Gaussian in feature space updates after the 10 scorings, using the aforementioned likelihood function in MCMC inference.

For each of the last 5 songs, we compare the feature vector of each song against the average value for each feature in the multivariate Gaussian. The function to calculate the score our model expects to assign is as follows:

$$\text{Expected Score} = 10e^{-c\vec{\Delta}\cdot\vec{\Delta}}$$

In the end, we are able to calculate the score our model thinks that the user will assign to a given song, based on the user's multivariate distribution of preferences and the given song's feature vector. Note that this is the same function our model uses to update in between samples, adjusting for the difference in actual assigned score and expected score. An example is as follows (using the first 10 songs as training data for the model):

Song Number	User Rating	Expected Score
1	8	-
2	3	-
3	6	-
4	7	-
5	2	-
6	5	-
7	6	-
8	7	-
9	3	-
10	1	-
11	7	7
12	5	6
13	4	3
14	3	5
15	8	6

Our model did not always perform this well. There are some songs with missing data points, which affected the way the model predicts the scoring of the song. For instance (again, using the first 10 songs as training data for the model):

Song Number	User Rating	Expected Score
1	4	-
2	6	-
3	2	-
4	1	-
5	5	-
6	7	-
7	6	-
8	3	-
9	6	-
10	8	-
11	6	7
12	2	1
13	9	1*
14	6	4
15	1	3

The starred value, in song 13, had no Tempo in the Echo Nest database, which caused the calculation for expected score to be lower than desired. On average, after validating our model on 25 samples, our model averaged a distance of 2.20 points away from the user assigned score. However, if we eliminate the mentioned data point where the song had no recorded Tempo, our predicted score was an average distance of 1.95 points away from the actual score. While there is room for improvement, we estimate some variation in likability to be expected, even among songs with similar musical features, which might consequently sound similar.

3.2. Recommending Music.

To find the closest song in the dataset, we first created a working subset of songs by randomly selecting 1000 songs from the million song dataset (the difficulty in searching the entire database prevented us from comparing a user's preferences to every song each user session). Then we equalized the axes in the feature space through the following approach:

$$\vec{\Delta} = \frac{\vec{p} - \vec{f}}{\vec{\sigma}}$$

Note that this is the same computation used to assess the distance between songs in our model, where $\vec{p}$ a sample from a user's multivariate Gaussian distribution of preferences, $\vec{f}$ is a song in the dataset, and $\vec{\sigma}$ is the standard deviation for each feature (operations are performed piece-wise). Thus, for each song in the 1000 song subset, we take $\delta = \vec{\Delta}_i \cdot \vec{\Delta}_i$. Then we find the closest songs from the calculated cosine distances in feature space, $min([\delta_1, \delta_2, ..., \delta_{1000}])$..

After training the model on 10 songs for each user, we sampled the adjusted multivariate Gaussian distribution and searched through our dataset for the song with the closet vector of features, using the mentioned distance formulation. We sampled the distribution 5 times to deliver each user 5 songs that our model evaluated to be the closest song in our working subset to the user's demonstrated preferences. We then asked the user to state whether or not he or she would listen to the song again, creating a standard for responses. After delivering 25 total songs to different users, the users delivered a total of 18 *yes* responses, indicating that the user would listen to the song again, and 7 total *no* responses, suggesting that the user did not like the recommended song.

We recognize that the standard for a *yes* response, in many cases, is low, which is why we measured the difference in expecting scoring in the previous section. However, an application of predictive scoring rests in song recommendation, and we wanted to explore how our model would perform in this context.

4. Data Storage and Processing

The logistics of collecting meaningful data from a large database can be difficult. The aforementioned million song database is made up of a hierarchal folder structure where the lowest level folders contain an hdf5 file for each song. The name of each hdf5 file is the track_id number. The track_id number is also used to navigate the folder structure. There are three levels of folders, each level containing folders 'A' through 'Z', and the third through fifth letters of the track_id number correspond to the folders containing that track. The million song database also provides a SQLite table containing a subset of the available metadata for each song. The SQLite table has data such as each song's title, artist, and year. The main purpose of the SQLite table is to help lookup songs in the database and provide the track_id number so the full song data can be found in the folder structure.

It is impractical to operate on the full database because of the large quantity of data stored for each song as well as the difficulty of accessing songs within the structure. To make the set of available songs more manageable in Python, we start by sampling a subset of 1000 songs. We use SQL queries to sample the full list of songs. We do this using the command "ORDER BY random() LIMIT 1000." Because "ORDER BY" creates a new table in memory to perform the sort, we only ask the query to return the title, track_id, and artist_name so that we limit the amount of memory needed to do the sort. We also tried sampling only the ROWID and then querying for the songs that match those ROWID's but did not see a significant speedup. Using our sampled list of track_id's, we can find the song file in the folder structure and extract our feature vector. We access the hdf5 files using a set of functions in the "hdf5_getters" file provided with the million song database. In order to reduce the number of accesses to the hdf5 files, we write our feature vectors to "song_subset.csv." This CSV file becomes our working database for the rest of the experiment. An example of a feature vector from the dataset is as follows:

[121.388, -6.062389200998751, 0.89383122996871, -11.293, 1997.0]

Where the features, in order, are Tempo, Timbre, Hotttnesss, Loudness, and Year. Tempo is in beats per minute, and the rest, other than Year, are in arbitrary units.

One difficulty we encountered while working with the million song database was the abundance of fringe, or less popular, genres. While we expect our model to be able to categorize these genres and avoid them for most people (assuming most people would not like fringe genres), there were enough in the database that a random sample of ten songs was likely to contain six or more uncommon pieces. This presents a problem in training our model because the space of possible songs is large

so it is easier to find similar songs to ones a user enjoys, rather than to find songs that the user is likely to enjoy based off songs he or she doesn't like. To obtain a sample of songs more relevant to potential users, we added a filter to our SQL query based on the "artist_hotttnesss", measure the Echo Nest uses to assess how popular an artist is.

Another difficulty we encountered was missing feature data within the database. For instance, we encountered songs without values for Tempo and Hotttnesss. For Hotttnesss, in the event of a missing value, we assigned the song a hotttnesss to be 0.5, the average value; however, for Tempo, we assigned the value to be 0. We thought that assigning Tempo any arbitrary value might have unwanted effects on updating the multivariate Gaussian, since Tempo seems to correlate more with Timbre and Loudness.

While the modeling and inference was executed in Webppl, a large portion of the code is contained in the Python language. Queuing songs that a user is likely to enjoy, based on the mentioned distance formula was completed in Python, as was randomly selecting songs to score in the first place. We have attached the Webppl code and the Python code can be requested. Some Python code was provided by Columbia University to access the database,[8] but a large portion had to be written for our specific usage to extract the appropriate information. The source code is available at: https://github.com/tfleech/6804-Music-Recommender.

5. Conclusion

Our model aims to capture a users preferences for particular musical features. In the end, our model was used to predict how a user would score a song, on a scale from 0 to 10, based on the user's previous scoring habits. We then attempted to use this metric to recommend songs in our dataset that were close in feature space to the user's preferences.

Our model performed well when a user responded positively to the given songs and not as well when the user was given a set of less desirable tracks. We believe our model could be improved by additional features in our feature vectors and through using a more robust dataset. As mentioned, some values in the dataset were missing, which disrupted our model at times. In addition, some songs in the dataset were not in English, and others contained no words. In these cases, it might have been difficult for a user to gauge to what extent he or she liked the song. We tried to eliminate bias by avoiding the use of a dataset only containing popular songs, but using a diverse dataset made it difficult to establish as user's preferences. After a training set of 10 songs, a user's preferences were often highly varied, making it difficult for our model to establish a confident multivariate distribution in feature space to represent the user's preferences. In addition, we think that using other metadata is probably extremely helpful. For instance, using collaborative filtering and natural language processing to aid in song suggestion seems like an effective approach for audio streaming companies, such as Spotify.

While many factors of the model could be improved with additional information, our formulation of distance between songs worked well, and our Gaussian model approximately described the data. From class, we learned that when dealing with many distributions, where an analytical solution is nearly impossible, it is more efficient to sample the distributions. With the five dimensional feature space, taking an analytical Bayesian approach became too difficult, and MCMC sampling was an efficient, and effective, inference technique to employ. We learned the importance of drift kernels, as well as the effects that burning samples and lagging between samples can have on your estimated posterior distribution. Moreover, the Webppl language made the inference easier to execute.

Overall, our model worked fairly well, and it provides a baseline for continued exploration into an efficient means to recommend songs to a user, based on the user's preferences.

6. References

[*] https://github.com/tfleech/6804-Music-Recommender (Our repository)

[1] https://en.wikipedia.org/wiki/Music-Genome-Project

[2] https://en.wikipedia.org/wiki/Songza

[3] https://www.wired.com/2015/12/songza-is-dead-but-it-lives-on-within-google-play-music/

[4] https://hackernoon.com/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe

[5] https://www.balabit.com/blog/category/unsupervised/

[6] https://news.spotify.com/us/2016/05/25/discover-weekly-reaches-nearly-5-billion-tracks-streamed-since-launch/

[7] https://help.netflix.com/en/node/9898

[8] https://labrosa.ee.columbia.edu/millionsong/

7. WebPPL Model

The main model in our WebPPL code is the following:

```
var exp_score = function(f, means) {
    return 10*Math.exp(-0.1*((f[0] - means[0])*(f[0] - means[0])/(sBPM*sBPM) +
        (f[1] - means[1])*(f[1] - means[1])/(sTimbre*sTimbre) +
        (f[2] - means[2])*(f[2] - means[2])/(sHotness*sHotness) +
        (f[3] - means[3])*(f[3] - means[3])/(sLoudness*sLoudness) +
        (f[4] - means[4])*(f[4] - means[4])/(sYear*sYear)))
}

var model = function() {
    var kernel = function(prevVal) {
        return MultivariateGaussian(mu: prevVal, cov: covs);
    }
    var userFavourite = sample(MultivariateGaussian(mu: means, cov: covs), driftKernel: kernel)
    var noise = sample(Exponential({a:1}))
    var ratingDistAt = function(fv) {
        return Gaussian({
            mu: exp_score(userFavourite.data, fv.data),
            sigma: 0.001
        })
    }
    observe statements
    return userFavourite
}
```