SMU Journal of Undergraduate Research

Volume 8 | Issue 2 Article 2

May 2024

Context Aware Music Recommendation and Playlist Generation

Elias Mann
Southern Methodist University, emann@smu.edu

Context Aware Music Recommendation and Playlist Generation

Cover Page Footnote

I would like to express my gratitude to Dr. Michael Hahsler, for his guidance and support throughout this research project. His insightful feedback and expertise have been instrumental in shaping the direction and focus of this work. I would also like the thank Dr. Eric Larson for his guidance and instruction in his machine learning course through which he allowed me to explore multi-task learning in this research as a project for his course.

Context Aware Music Recommendation and Playlist Generation

Elias Mann[1]

emann@smu.edu

Michael Hahsler[1]

ABSTRACT

There are many reasons people listen to music, and the type of music is largely determined by what the listener may be doing while they listen. For example, one may listen to one type of music while commuting, another while exercising, and yet another while relaxing. Without access to the physiological state of the user, current music recommendation methods rely on collaborative filtering - recommending music based on what other similar users listen to - and content based filtering - recommending songs based on their similarities to songs the user already prefers. With the increasing popularity of smart devices and activity trackers, physiological context can emerge as a new channel to inform music recommendations. We propose deep learning solutions for context aware recommendation and playlist generation. Specifically, we use variational autoencoders (VAEs) to create a song embedding. We then explore multi-task multi-layer perceptrons (MLPs) and Gaussian mixture models to recommend songs based on context. We generate artificial user data to train and test our models in online learning and supervised learning settings.

1. INTRODUCTION

As smart devices and activity trackers become more available and affordable, more people purchase these products which constantly monitor their physiological state. We believe that music recommendation systems can be improved and personalized by leveraging the increasing prevalence of activity trackers. In general people listen to different types of music for different activities [1]. We believe that by recommending the right music for these activities, we can build a recommendation system that can enhance the experiences for the user.

In this paper, we explore two different methods for recommending music based on context. While we experiment with the context groups running, commuting, and relaxing, context can be expressed in a variety of ways including the user's physiological state or location for example.

For our analysis, we use a subset of the Million Spotify Playlist Dataset [2]. This subset consists of 32,200 songs retrieved using the Spotify API. For each song we retrieve metadata - genres, popularity, artist etc. - and song features - tempo, key, loudness etc. - from the Spotify API. All features and metadata were created by Spotify [3].

We use a VAE [4] to create a regularized song feature embedding. We use genre-based clustering within that embedding to generate artificial users with specific preferences for different activities. For our first recommendation method, we create a multi-task neural network to predict during which activities a user would listen to given song. For our second method, we created an online learning environment in which we created a Gaussian mixture model [5] to learn user preferences and create online task-specific song recommendations.

2. BACKGROUND

2.1 Recommendation Systems

Recommendation systems filter information in order to personalize which content is presented to a user at a given time [6]. Most state-of-the-art music recommender systems use a combination of *collaborative filtering* and *content based filtering* [7].

In the context of music recommendation systems, *collaborative filtering* [8, 9] is a process that uses the relationships between multiple users and their listening preferences to make recommendations to individual users [10]. For example, if user 1 and user 2 have similar listening histories, user 1 may be recommended a song that user 2 likes that user 1 has not yet heard. This technique is implemented in varying levels of complexity.

Another umbrella term for recommendation strategies is *content-based filtering* [11, 12]. This refers to a recommendation system which uses features from the songs in a user's listening history in order to recommend other, similar songs. Similarity can be represented in many layers of abstraction from raw audio signals to similarities between lyrics, to song metadata. The popular music streaming service Spotify uses a combination of these methods to personalize user recommendation [3].

Another increasingly prevalent type of recommendation system is *context-based recommendation* [13, 14]. These systems further personalize recommendations by considering contextual factors like location, weather, and surroundings. Context can be provided in a variety of methods including using external internet-of-things (IOT) devices [15] like smart watches and activity trackers.

[1] Department of Computer Science, Southern Methodist University

2.2 *Embeddings*

When working with high dimensional data, it is common to use an embedding to create lower dimensional, dense vector representations of the original data. This can be useful since working directly with high dimensional data requires larger amounts of computation, which can lead to "the curse of dimensionality"; a phenomenon in machine learning algorithms where performance deteriorates due to high dimensionality [16], and may not capture meaningful relationships in the data.

When the data is compressed to lower dimensions, the process can reduce noise, and redundant and irrelevant information in the data. In doing so, a proper embedding highlights meaningful patterns and relationships in the data, which can improve model performance and generalizability [17].

Common uses for embeddings can be found in natural language processing (NLP) with the popular GloVe [18] and Word2Vec [19] embeddings which are used to represent words in a continuous vector space. These vectors represent the semantic relationships between words and are the foundation of many NLP models.

We create a song embedding using a variational autoencoder to train our recommendation and playlist generation models.

2.3 *Variational Auto Encoders*

A variational autoencoder (VAE) [4, 20] is an encoder-decoder model. The encoder learns to map data from a high dimensional input space to a low dimensional latent space, and the corresponding decoder maps the data from the low dimensional latent space back to the high dimensional input space. Unlike traditional autoencoders, VAEs use variational inference to force a regularized distribution on the latent space. This is done by modifying the loss function to include a Kullback-Leibler (KL) divergence penalty term, and by employing a sampling layer to generate new data from the learned latent space.

The KL divergence term is a measure of how different the latent space distribution is from a standard normal distribution. The KL divergence term is minimized by the VAE so the latent space distribution is as close to a standard normal distribution as possible.

The VAE is trained by minimizing the sum of the reconstruction loss and the KL divergence term. In the simplest methods, the reconstruction loss can be defined by mean squared error.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\text{KL}(q_\phi(z|x)||p(z)) = -\frac{1}{2}\sum_{j=1}^{J}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right)$$

$$\text{VAE Loss} = \text{MSE} + \text{KL}$$

It is also common to use binary cross entropy (BCE) for reconstruction loss and add a small scaling factor λ to the KL divergence term. We also tested our VAE using this method but found better results with MSE. This could be because we had more continuous than discrete feature values, MSE tends to perform better for continuous data while BCE tends to perform better for discrete data.

Another way that smoothness in the latent space is enforced in the VAE is through the sampling layer. This allows the models to generate new samples that are similar to the ones that it has already been trained on. During the sampling process a noise variable is added which is randomly drawn from a Gaussian distribution. Samples are generated by mapping the variable to the mean and variance of the learned distribution of the latent space. This process is called the reparameterization trick, and makes sampling differentiable so that the model can be optimized using gradient descent [4]. This enforces smoothness by constraining the latent vectors to be near the origin and exhibit a Gaussian distribution.

Together, the KL-divergence penalty term and sampling layer force the learned latent space to be smooth, regular, and distances between points in the latent space to be meaningful.

2.4 *Multi-Task Learning*

Multi-task learning [21] is an approach where a single model is trained to perform multiple related tasks simultaneously, rather than training separate models for each task. This approach is based on the intuition that tasks often share common features or information; by learning them jointly, the model can exploit these shared characteristics to improve performance on all tasks.

In the context of music recommendation, this learning strategy can be utilized to ameliorate the cold start problem, an issue with content-based recommendation systems [22]. If each task represents a different user, weights for the branches of a new user can be initialized as an average of the weights from the other users. This would be an example of introducing a "warm start" to the model.

2.5 *Gaussian Mixture Models*

A Gaussian mixture model (GMM) [5] is a probabilistic model that assumes that the observed data is generated from a mixture of several Gaussian distributions, each with their own mean and covariance matrix. The PDF for a Gaussian mixture model is as follows

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

where x is the observed data, π_k is the mixing coefficient for the kth Gaussian distribution, and $\mathcal{N}(x|\mu_k, \Sigma_k)$ is the multivariate Gaussian probability density function with mean vector μ_k and covariance matrix Σ_k.

In this paper, the user preferences are modeled by a Gaussian mixture with each activity being a component k. Each distribution has full covariance and the component is explicitly defined at training and inference.

2.6 *Online Learning*

Online learning [23] is a process in which a model is trained on real-time data as it becomes available, continuously adapting the model. This is in contrast with

Feature/Metadata	Explanation	Type
acousticness	A confidence measure of whether the track is acoustic.	float
energy	A perceptual measure of intensity and activity.	float
danceability	A measure of how suitable a track is for dancing.	float
instrumentalness	A likelihood measure of whether the track contains no vocals.	float
key	The key of the track. Encoded with pitch class notation [25].	integer
liveness	A likelihood measure of whether the track was performed with an audience.	float
mode	Indicates if the track is major or minor.	integer
speechiness	A measure of whether the track contains spoken (not sung) words.	float
tempo	Estimated beats per minute of a track.	float
valence	A measure of the musical positiveness conveyed by a track (happy or sad)	float
artist popularity	Popularity of the artist 0 to 100.	integer
genre	List or genres associated with artist.	array of strings
track popularity	Popularity of the track 0 to 100.	integer

Table 1: Descriptions of track features and metadata retrieved from Spotify API, derived from its documentation [26].

batch learning in which the model is trained in batches of data after it has all been collected. Online learning is useful for models that need to adapt to changes in the distribution of data, and continuously improve the model as new data becomes available. This is especially useful in recommendation systems, speech recognition, and autonomous vehicles [24].

3. METHOD

3.1 Dataset

Our data was a 32,200 song subset for the Million Spotify Playlist Dataset [2]. This dataset consists of user generated playlists on Spotify, each with a list of songs and identifying information about each song. The dataset was publicly released in 2018 as part of a recommendation system challenge and is available for research and noncommercial use.

The information we used were the track id, and artist id. With these identifiers, we used Spotify API to retrieve features and metadata about each song. Song features were created by Spotify through their proprietary music information retrieval models [3]. The features and metadata we used are detailed in Table 1.

We simplified genres by only assigning the first genre in the genre list to each track. Additionally, we only maintain the top 25 genres, which account for 78% of our data. The rest of the genres we labeled as 'other'. All of the listed metadata and features except for genre were used to create our song embedding. Songs with similar features sound similar [3] and songs in the same genre also tend to sound similar. We used the genre to validate that our embedding created meaningful spatial representations of song similarity. Songs in the same genre should tend to be closer to one another than to songs from different genres.

3.2 Song Embedding

In order to create our song embedding, we constructed a (VAE) using mean-squared error and KL divergence for the loss function. After encoding our categorical features, we expanded our 12 input features to 24 dimensions. The VAE scales feature representations down to a 12 dimensional latent space. We chose to scale down to half of the original dimensions in order to preserve information from the original features, while still allowing the model to be deep enough to learn complex relationships in the data. The encoder and decoder each had 3 dense layers. In the encoder the size of each subsequent layer was scaled down by a factor of approximately 0.75. In the decoder, the size of each subsequent layer was scaled up by a factor of approximately 1.3. Hidden layers used ReLU [27] activation functions and the decoder output is tanh [28] activation function.

3.3 Multi-Task Model

The goal for our multi-task model wad to generate playlists for different activities given the user's listening history. The input was a song and the model predicted which activity class it belonged to. Multiple activities could be predicted for a given song. In this model, we started by retrieving the 12 dimensional latent space vector for a given song. While the rest of the model was trainable, the encoder was not. The general branch and shared dense layer were trained first on all of the users in a supervised setting. The shared layer had 12 units, the same as the latent space, and acted as a trainable embedding layer for the model. There were two middle layers in the general model with sizes of 9 and 7 and ReLU [27] activation. The output later had 3 units with sigmoid [29] activation. The user branches, which were identical to the general model in architecture, were initialized with the weights of the general branch. During the training for the rest of the model, the general branch was removed. In each training step, the model alternated training between each user branch as not to over-fit the model to one user. All models shared an early dense layer, the intuition for which was to allow the model to learn shared information across all users, while also allowing each user branch to learn information specific to their listening preferences. By having a shared dense layer, the model could learn common features that are important for all users. At the same time, each user branch could learn personalized features that were unique to their individual preferences. This approach

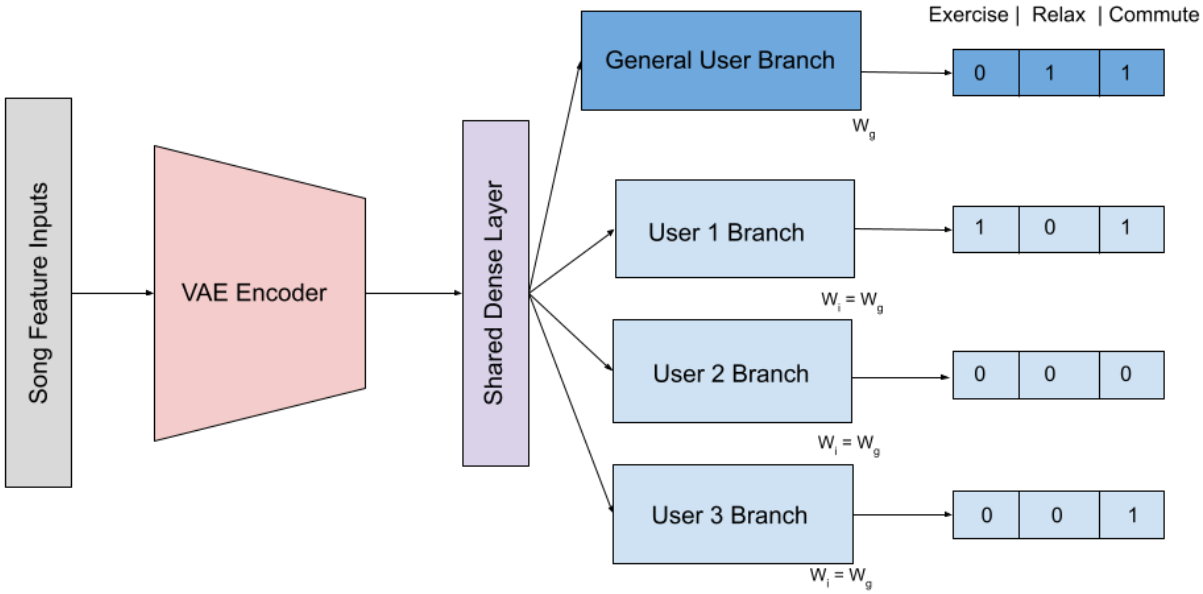

Fig. 1: Multi-task recommendation architecture. Output is a sigmoid activation function with 3 output units each corresponding to the probability of a song existing in a given activity preference. Notice that one song can be recommended for multiple activities. W_g denotes the weights of the general branch, W_i denotes the initial weights for a user branch.

allowed the model to balance between generalization and personalization, which is important for creating a successful recommendation system that can generalize to a diverse set of users.

3.4 Gaussian Fit Model

The Gaussian fit model was trained in a online learning environment. It was initialized with 4 preferences for the user for each activity. The model then batch fitted a Gaussian distribution with full covariance to the initial points.

Given a batch of data $X \in \mathbb{R}^{m \times n}$, where m is the number of samples and n is the dimensionality of each sample, batch fit computes the maximum likelihood estimates of the parameters of a Gaussian component j with full covariance matrix: The mean vector $\hat{x}_j \in \mathbb{R}^n$ is estimated as:

$$\hat{x}_j = \frac{1}{m}\sum_{i=1}^{m} x_i$$

where x_i is the i-th sample in X. The covariance matrix $Q_j \in \mathbb{R}^{n \times n}$ is estimated as

$$Q_j = \frac{1}{m}\sum_{i=1}^{m}(x_i - \hat{x}_j)(x_i - \hat{x}_j)^T$$

At inference time, the model randomly picked a song that fell within radius r away from the mean of that Gaussian in the latent space for recommendation. For our experiment we chose r to be 1 standard deviation of the Gaussian distribution in order to include characteristic songs for that distribution. It then proposed the song to the user. If the user rejected the song, then the model recommended a new song. Previously recommended songs were not re-recommended within the same session. If the song was accepted, the model online-fitted the distribution to include the newly accepted song.

Given a new observation $x \in \mathbb{R}^n$, the online-fit updates the running mean μ:

$$\hat{x}_j \leftarrow \hat{x}_j + \alpha_\mu(x - \hat{x}_j)$$

where the scaling factor $\alpha_\mu = \frac{1}{N}$. N Is the total number of observations x that the distribution has been fitted to, including those that were fit with batch fit. The function updates the running covariance matrix Q_j:

$$Q_j \leftarrow Q_j + \alpha_\sigma[(x - \hat{x}_j) \otimes (x - \hat{x}_j) - Q_j]/N$$

where $\otimes$ denotes outer product, $/$ denotes element-wise division and the scaling factor $\alpha_\sigma = \alpha_\mu(1 - \alpha_\mu)$.

When a user accepted a new song the next song could picked as a "nearby song" within the distribution. This means that the next song choice could be biased towards an area in the latent space that was near to the previously accepted recommendation. We dd this by defining a new distribution with the mean at the location in the latent space of the previously accepted song, and the same covariance matrix as the overall preference distribution. The new distribution was then scaled down in magnitude by a tunable parameter that defaults to 10%. The next recommendation was chosen from the subset of songs that existed in the area that was within one standard deviation from the mean of the new distribution and within one standard deviation from the mean of the overall preference distribution.

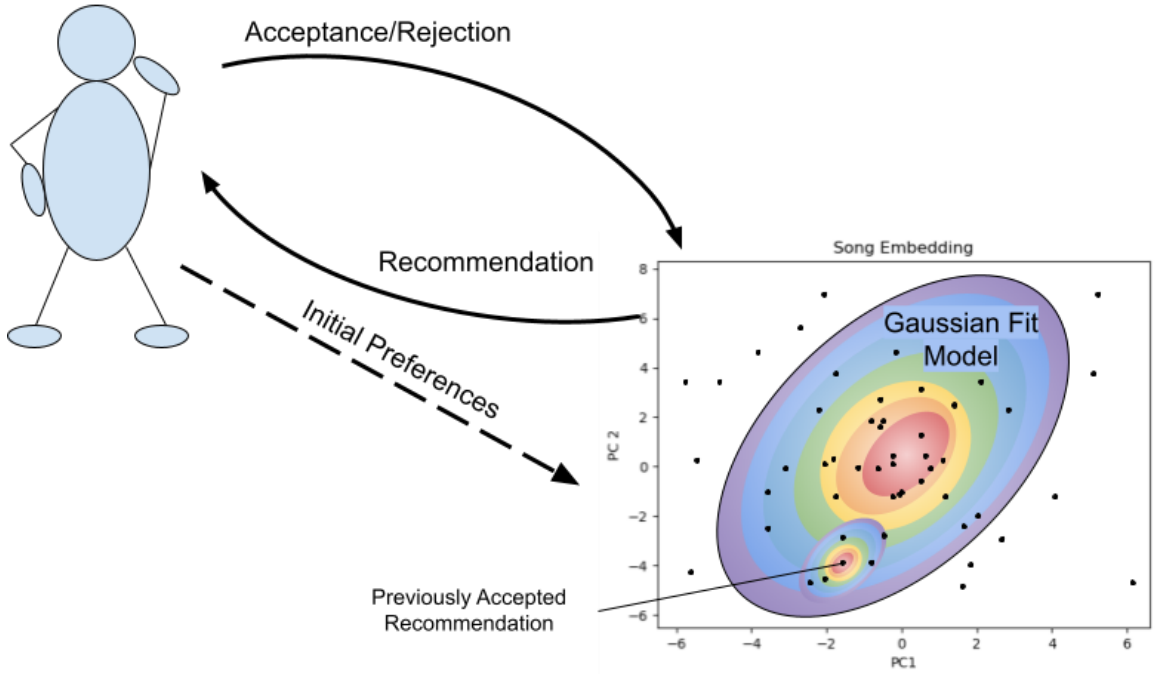

Fig. 2: Online learning environment for Gaussian mixture model. Fitting a Gaussian distribution with full covariance to user preferences in latent space. Notice that when the previous recommendation was correct, the model may sample from within a similar scaled-down Gaussian distribution to enforce closeness to the previous song. The actual latent space is 12 dimensional, yet we are depicting 2 principal components in this figure.

In order to avoid over fitting to one area in the latent space, the next song recommendation was not always a "nearby song". Every *b* recommendations, the next song was randomly sampled from the overall preference distribution without nearby bias. *b* was a tuneable parameter. Since the model was actively learning, this parameter encouraged the mean and covariance of the model to better represent that of the users true preferences, and mitigated optimizing to small local areas of the latent space.

The intuition behind choosing a nearby song to the previously accepted song was to capture the preferences of the current listening session. Additionally, just because a user liked listening to two different songs for exercising, that does not necessarily mean that the user wants to hear them consecutively. Choosing a "nearby song" can facilitate smoother transitions between songs in a session. An intuition behind abandoning closeness after successfully recommending *b* songs was that the user gets tired of listening to songs that sound similar and want to hear something new.

4. EXPERIMENT

4.1 Generating Artificial Users

Without access to a large user base or test subjects, we created artificial users in order to efficiently train and test our models. To create these users, we used genre-based clustering. Since songs in the same genre have higher probabilities of sounding similar, we believe this is a convenient way to create realistic a listening preference. We initially compiled a list of genres that would be characteristic genres to listen to for exercising, commuting, and relaxing. These lists of genres did have some overlapping genres. The listening preference for each activity for each user was represented by a randomly chosen genre that represented the activity. In order to create the clusters and populate the listening preferences, we found the centroid of a given genre and determine all of the songs which were within a given radius from that centroid - for our experiments, we chose one standard of deviation as the radius. By selecting all of these songs, we are left with songs that sound like a characteristic song of the genre, even if it was not explicitly labeled as that genre. Next we introduced randomness which we implemented in two different ways. The first was to choose a percentage of the songs within the radius, remove them from the preferred songs, then randomly sample the same number of songs from the rest of the latent space and add them to the preferences. The second method was to define a second radius in the latent space - 2 standard deviations from the mean in our experiments - and the new randomly sampled songs came from the area in the latent space between the radii. Both methods of introducing randomness were attempts to more accurately simulate a real user's preferences. The second method intuits that the outlier songs will still be somewhat near the characteristic centroid song, not on the opposite side of the latent space.

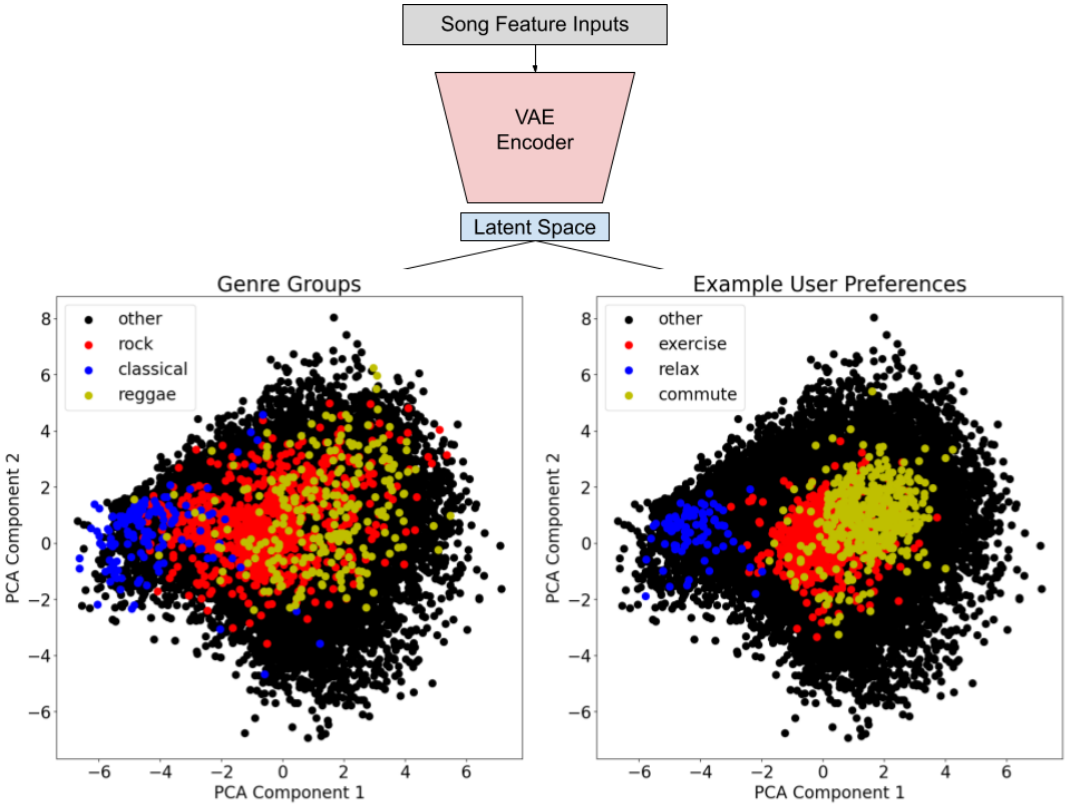

Fig. 3: Left: Representation of different genres in the latent space. Right: Preferences for an artificial user with a radius of 1 standard deviation, an outer radius of 2 standard deviations and randomness of 25%.

4.2 Multi-Task Training

We first trained our multi-task model on 3 users with a centroid radius of 1 and 10% randomness. Then we trained them on similar users with 20% randomness. These users did not include an outer radius so the random sampled were from the entire latent space. As detailed in section 3.3, we first train a general branch to initialize the weights of the individual user branch, remove the general branch after weight initialization, then train the multi-task model to convergence.

4.3 Gaussian Fit Training

We trained our Gaussian fit model in an online learning environment. In our environment, the user initially gives n songs that they prefer for each genre, which we batch fit our model to. This was the smallest amount of initial samples required to reliably be able to sample real songs from the latent space. After a song is recommended, the user accepts or rejects the song. As detailed in section 3.4, the model fits to accepted songs, and may enforce the next recommendation to be close to the previously accepted song. We trained models on 2 types of users: a user whose preferences are randomly sampled from the entire latent space, and on a user with an inner radius of 1 standard deviation, an outer radius of 2 standard deviations, and 20% randomness. The completely random user was tested with a model that does not select nearby recommendations. The second user was tested with 3 different model configurations: 1 that does not recommend nearby samples, 1 that can recommend up to 5 consecutive nearby samples, and 1 that always recommends a nearby sample if the previous sample was correct. All models were trained with an initial 4 random samples songs for each activity. After batch fitting on these samples, we ran our online learning environment for 3000 episodes, switching between activities every 10 episodes as a rudimentary way of simulating users moving between environments.

5. RESULTS

We found that our multi-task model is able to accurately predict preferences for 3 different users across 3 different listening contexts with the low AUC of 0.86 and high of 0.96 a across all users and contexts. We also found that the multi-task nature of our model with individual branches for each user does allow for a better representation of individual user preferences. This is evident from our individual user branch scores being greater than or equal to the scores of the general branch across all contexts. The AUC scores for each user across all contexts decrease by about 0.05 when the user's random preferences are increased from 10% to 20%. The AUC scores for users with 20% random preferences were still high with the lowest being 0.86 and highest being 0.89. This shows that our model can generalize well to users with more sporadic listening preferences.

We also found our best Gaussian fit model recommends songs in our listening simulation environment with an average of 92.6% accuracy across all listening contexts. Averaging over 100 simulations of 1000 episodes

Model	User	Exercise	Relax	Commute
General Branch 10% Random		0.90	0.88	0.90
Individual Branches 10% Random				
	User 1	0.93	0.96	0.94
	User 2	0.94	0.91	0.94
	User 3	0.93	0.92	0.90
Individual Branches 20% Random				
	User 1	0.89	0.88	0.86
	User 2	0.88	0.88	0.87
	User 3	0.87	0.88	0.89

Table 2: AUC scores for each branch with a 10% random user, and AUC scores for individual branches with a 20% random user.

on different users, the model had an accuracy of 97% for exercise context, 85.9% for relax context, and 94.9% for commute context.

Interestingly, the highest performing model is the one which can recommend unlimited consecutive "nearby songs". We expected this model to over-fit to small areas in the latent space, and therefore be unable to effectively model the user preference distribution across contexts. The model which we designed to mitigate over-fitting by only recommending 5 consecutive "nearby songs" performed worse than the unlimited model with an average accuracy of 84.6% across all contexts. Our model which does not recommend "nearby songs" performed the worst with an average accuracy of 64.4% across all contexts.

5.1 Multi-Task Model

For our AUC scores and ROC curves [30], true positives are when the model predicts a greater than 50% chance that a song belongs to a given activity and that song is indeed a preference for the given user. A false positive is when the model predicts a greater than 50% chance that a song belongs to a given activity, and it is not a preference for the user.

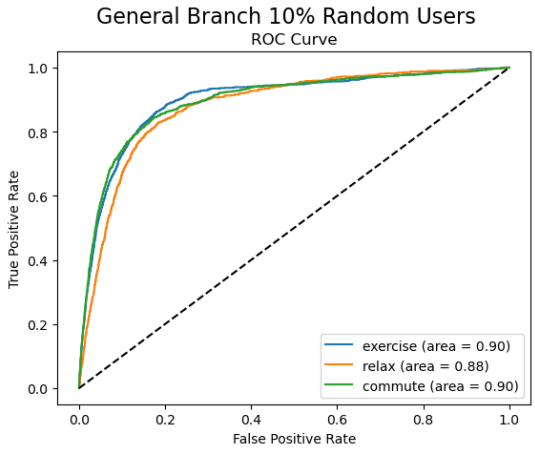

Fig. 4: ROC curve for the general branch of the multi-task model. AUC scores: exercise = 0.90, relax = 0.88, commute = 0.90

We found that the AUC scores for the individual user branches exceeded those of the general model for relax and commute, but were the same for exercise (see Figure 5). The individual users also had slightly varying scores. We expected the user branches to have better results than the general branches since they were designed to optimize to their respective users. We believe inconsistencies between the individual models could be due to some preferences being more difficult to learn than others. Since the genres which represent activities are chosen at random, activities with lower AUC scores could have fewer examples to learn from.

We found that when predicting users with 20% randomness instead of 10% randomness, that while our model scores did decrease by about 0.05 on average, the AUC scores were still high (see Figure 6). This shows that the model can generalize well to users with less consistent preferences.

5.2 Gaussian Fit Model

Accuracy is defined by the total number of accepted recommendations divided by the total number of recommendations made during the simulation.

We validated our model by conducting four different experiments. The first was to test the Gaussian fit model on completely random data. As expected accuracy was low, this establishes a baseline to compare models on no-random users. This experiment also shows that the Gaussian bias of our model does not allow it to over-fit to random noise.

Next we generated a user, and tested a model that does not make nearby recommendations. This model fits an individual Gaussian distribution to each listening context. While the performance of this model was still able to adequately make 66-69% correct predictions depending on the context, this model performed the worst out of the models fit to non-random users.

Our next model used a nearby recommendation method where if the previously recommended song was accepted, it would choose a "nearby song". However, after 5 consecutive nearby songs, the model again sampled from the overall preference distribution. We believed that this would be the best method as it would be more likely to focus on areas of the latent space where preferences have been found, but would not over-fit to that area. This model outperformed

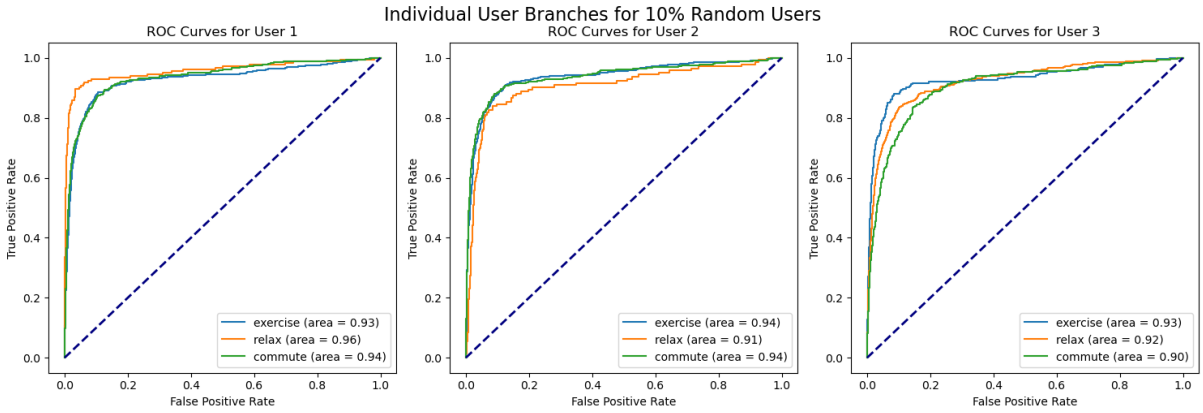

Fig. 5: ROC curve for the user branches of the multi-task model with 10% randomness. User 1 AUC scores: exercise = 0.93, relax = 0.96, commute = 0.94. User 2 AUC scores: exercise = 0.94, relax = 0.91, commute = 0.94. User 3 AUC scores: exercise = 0.93, relax = 0.92, commute = 0.90.

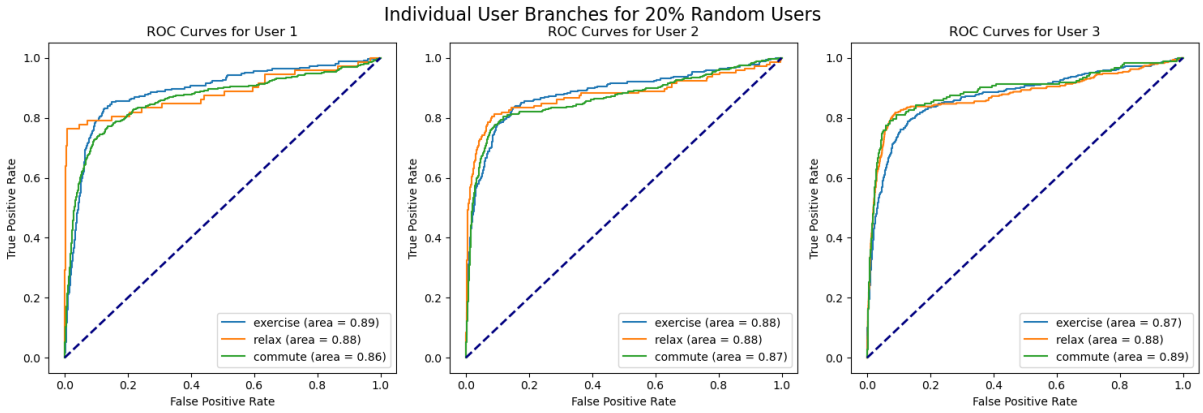

Fig. 6: ROC curve for the user branches of the multi-task model with 20% randomness. User 0 AUC scores: exercise = 0.89, relax = 0.88, commute = 0.86. User 1 AUC scores: exercise = 0.88, relax = 0.88, commute = 0.87. User 2 AUC scores: exercise = 0.87, relax = 0.88, commute = 0.89.

the random model drastically and it outperformed the model without nearby recommendations by about 20% across all contexts. However this model did not outperform the model without a limit to nearby preferences.

The final model will always pick a "nearby song" if the previous recommendation was accepted. Despite our worries about over-fitting, this model performed the best, exceeding the accuracy of the model that limits nearby recommendations to 5 by about 10% across all listening contexts.

We tested each model on 100 different users with 20% randomness, an inner radius of 1 standard deviation and an outer radius of 2 standard deviations to obtain average accuracy for each configuration. Interestingly, the models trained on non-random users performed about 10% worse on average for the relax context than they did for the other contexts. While we expected slight variation between contexts, we did not expect to see this significant of a variation averaged over 100 different users. This could be because the centroids of the genres in the relax context tend to be in a sparser area of the latent space.

In Figures 7-9 we generated a single user and visualized how each of our 3 models updated their means during a simulation of 1000 episodes.

In Figure 7, we see the means of the model which does not recommend any "nearby songs" to a previously accepted recommendation. In this example the mean only appears for the exercise context. It is likely that this is because the model is making more correct recommendations, allowing it to more accurately fit to the distribution of the data.

Even though the initial and final means for the commute context were nearby the true mean, the component still had low accuracy. The relax component had comparable accuracy to the commute component even though the initial and final means in the relax component were far from the true mean.

In Figure 8 we see the means of the model that can recommend 5 consecutive "nearby songs". In this example, we see that final means are far from the true means in all three contexts, and they do not seem to move far from their initial positions in the latent space. Since the models have decent accuracy scores, we know that correct

recommendations are being made. This may indicate that the model is over-fitting to its initial area in the latent space.

Model	Exercise	Relax	Commute
Random User No Nearby	3.2%	2.7%	2.8%
No Nearby	68.1%	58.9%	66.1%
5 Nearby	89.7%	77.9%	86.3%
All Nearby	97.0%	85.9%	94.9%

Table 3: Average accuracy over 100 users for each activity. Each activity runs for 1000 episodes. The first row is a model fit on users with completely random preferences. The other rows are models fit on a users with 20% random preferences with different nearby recommendation limits.

It could be argued that this is because the initial mean is already nearby the true mean in the exercise and commute components; however, we can see in the relax component that the initial and final means are far from the true mean.

However the means of the exercise and commute contexts are nearby the true mean, so they may be adequately representing the data.

These plots show that the correlation between proximity to true mean and accuracy is complex and not easy to interpret from the plots. This may be a result of the 2 dimensional PCA plot being unable adequately represent the true nature of the 12 dimensional space.

In Figure 9 we see the means of the model that can recommend unlimited consecutive "nearby songs". In these plots, we have the highest accuracy out of all of our models. However we don't see a clear trend of predicted means approaching the true mean. Perhaps the model is over fitting to a certain area of the latent space and there are enough positive examples in that area to yield a high score. The final means do seem to travel farther from the initial means in this example as well.

We see similar trends with Figure 8 where the predicted means of the relax component biases towards a specific area in the latent space which is far from the true mean. We also see that proximity of predicted means to the true mean does not necessarily correlate to higher accuracy.

6. CONCLUSIONS

In this work, we have demonstrated performant methods of modeling user song preferences across different user listening contexts. We have created methods for a supervised learning setting using a multi-task neural network, and in an online learning setting with our Gaussian fit model. Both models use a song embedding created by a VAE.

Our multi-task neural network has less bias and is more generalizeable than our Gaussian fit model, but must be trained on a larger set of user listening history. With enough data, this model can be used to predict user song preferences with high AUC scores of up to .96. This model classifies whether or not a song would be preferred by a user in a given context, and could be used to assign songs to the appropriate playlists for different activities.

Since our Gaussian fit model has more bias, it is better suited to recommend songs when the listening history is limited. This Gaussian bias is well suited recommend songs based on distances in an embedding space created by our VAE because the VAE enforces a smoothness and regularization in latent space. This model recommends songs and updates parameters based on whether or not a song is accepted for a given listening context. A different Gaussian distribution with full covariance is fit to each listening context.

We also found that the Gaussian fit model makes better recommendations when it biases subsequent recommendations to areas nearby a previously accepted recommendation in the latent space. This model can recommend songs with an average accuracy of up to 97%.

If used in tandem, the Gaussian fit model could be used to recommend songs and create a listening history. When the listening history is large enough, the multitask neural network can be used to create personalized playlists for different listening contexts.

It is important to address the limitations involved in this experiment regarding the use of artificially generated users. The methods used to train and test our models are only constructive if our artificial users can accurately model a true user. We designed the artificial users and constructed a model that is optimized to solve it. However, we do believe our methods were stringent and our artificially generated users do represent a useful representation of user's listening habits for different contexts.

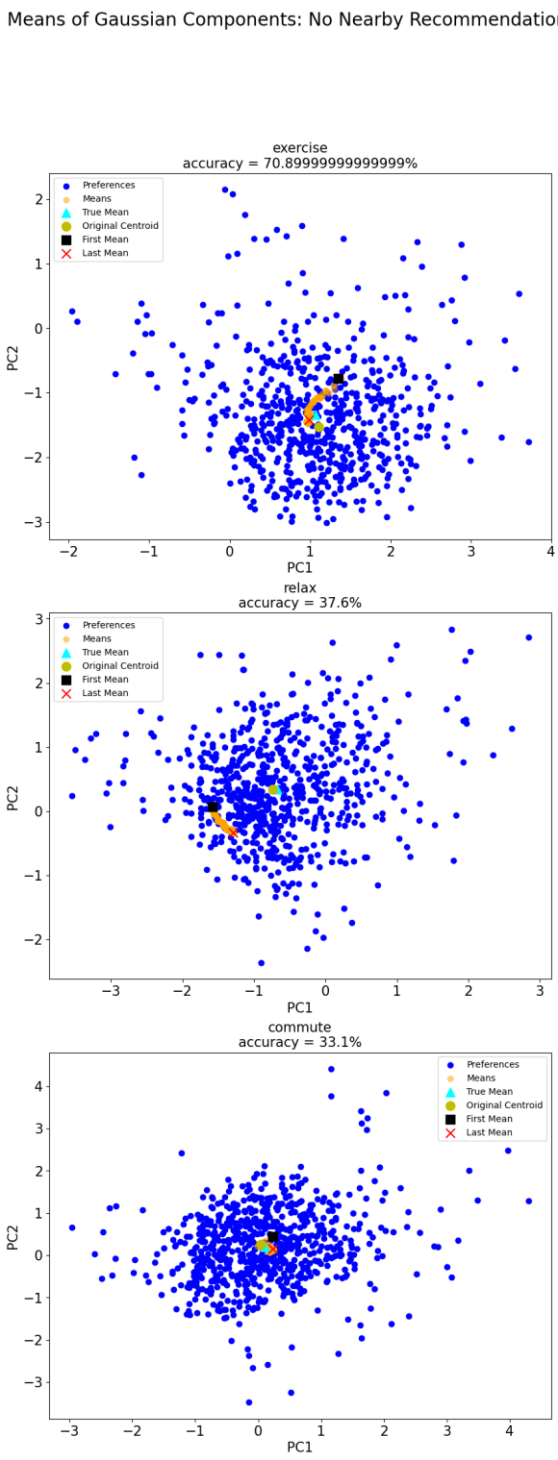

Fig. 7: Principal components plot with the means of a model that does not recommend “nearby songs”. The cyan triangle depicts the true mean of the activity preferences, the yellow circle depicts the mean of the preference before adding noise, the black square depicts the initial mean of the model after batch fitting, the red ’X’ depicts the final mean of the Gaussian, and the orange dots represent the path that the mean of the Gaussian fit takes as it travels from first to last mean. Blue dots are ground truth user preferences.

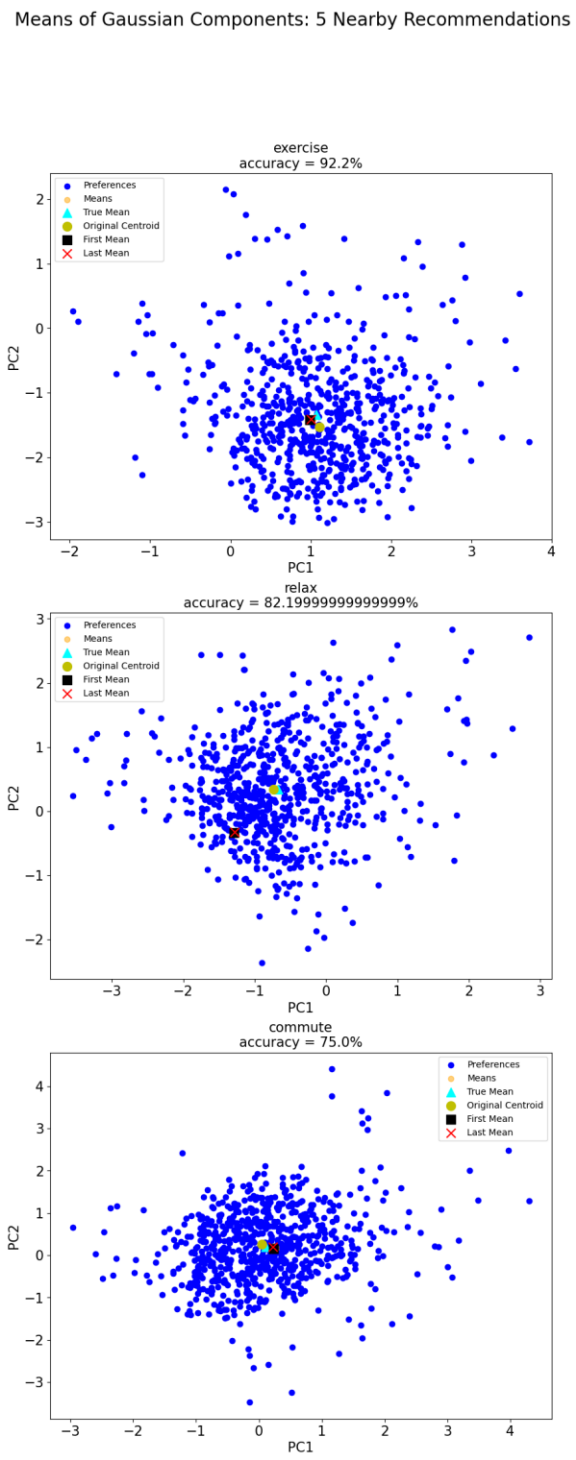

Fig. 8: Principal components of the means of a model that recommends up to 5 consecutive “nearby songs”. The cyan triangle depicts the true mean of the activity preferences, the yellow circle depicts the mean of the preference before adding noise, the black square depicts the initial mean of the model after batch fitting, the red ’X’ depicts the final mean of the Gaussian, and the orange dots represent the path that the mean of the Gaussian fit takes as it travels from first to last mean. Blue dots are ground truth user preferences.

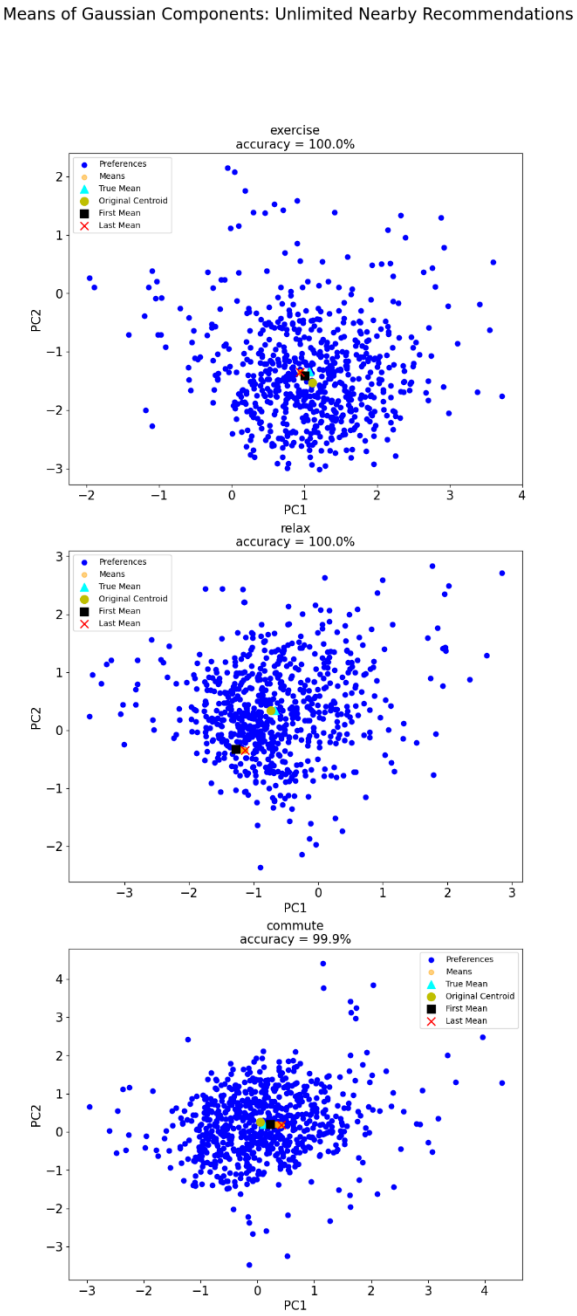

Fig. 9: Principal components of the means of the model can recommend unlimited consecutive "nearby songs". The cyan triangle depicts the true mean of the activity preferences, the yellow circle depicts the mean of the preference before adding noise, the black square depicts the initial mean of the model after batch fitting, the red 'X' depicts the final mean of the Gaussian, and the orange dots represent the path that the mean of the Gaussian fit takes as it travels from first to last mean. Blue dots are ground truth user preferences.

7. FUTURE WORK

To expand on this work, we could improve on our song embedding by experimenting with different types of encoders. We use a simple VAE with mean squared error as reconstruction loss because our data contained more continuous than categorical variables. However using a method like Gaussian Bernoulli Variational Auto Encoder (GB-VAE) [31], we could compute a binary cross entropy reconstruction loss for one-hot encoded categorical variables, and mean squared error reconstruction loss for continuous variables. Other encoding methods like an Adversarial Auto-Encoder (AAE) [32] or Adversarial Latent Auto-Encoder (ALAE) [33] would be able to better enforce a desired distribution on the song embedding.

We could improve our artificial users by representing each activity preference as a random mixture of *n* genres pertaining to the activity. This could result in a better representation of actual user preferences and be left with more irregularly distributed preferences.

We may also improve our Gaussian fit model by representing the overall preference distributions for each context as Gaussian mixture. By using Variational Bayesian Gaussian Mixture Model [34] we could learn the number of Gaussian components instead of setting the number as a fixed parameter.

We believe that recommending music to listeners based on their current activities and listening contexts can enhance the user's listening experience [1]. We envision our models recommending music customized to tasks like exercising, studying, relaxing or commuting. As activity tracking technology becomes more accessible and unobtrusive, our methods can lay the groundwork for the next leap in music recommendation.

8. ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Michael Hahsler, for his guidance and support throughout this research project. His insightful feedback and expertise have been instrumental in shaping the direction and focus of this work.

I would also like the thank Dr. Eric Larson for his guidance and instruction in his advanced deep learning course through which he allowed me to explore multi-task learning for this research

9. REFERENCES

[1] Lonsdale, A.J., North, A.C.: Why do we listen to music? a uses and gratifications analysis. British Journal of Psychology **102**(1), 108–134 (2011) https://doi.org/ 10.1348/000712610x506831

[2] Chen, C.-W., Lamere, P., Schedl, M., Zamani, H.: Recsys challenge 2018: Automatic music playlist continuation. Proceedings of the 12th ACM Conference on Recommender Systems (2018) https://doi.org/10.1145/3240323.3240342

[3] Jacobson, K., Murali, V., Newett, E., Whitman, B., Yon, R.: Music personalization at spotify. RecSys '16. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2959100.2959120

[4] Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes (2022). https:// doi.org/10.48550/arXiv.1312.6114

[5] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em - algorithm plus discussions on the paper. (1977) https://www.ece.iastate.edu/~namrata/EE527_Spring08/Dempster77.pdf

[6] Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. Springer (1970). https://link.springer.com/chapter/10.1007/ 978-0-387-85820-3 1

[7] Basilico, J., Hofmann, T.: Unifying collaborative and content-based filtering. In: Proceedings of the Twenty-First International Conference on Machine Learning. ICML '04, p. 9. Association for Computing Machinery, New York, NY, USA (2004). https://doi.org/10.1145/1015330.1015394

[8] Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Commun. ACM **35**(12), 61–70 (1992) https://doi.org/10.1145/138859.138867

[9] Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in artificial intelligence **2009** (2009) https://doi.org/10.1155/2009/421425

[10] Shakirova, E.: Collaborative filtering for music recommender system. In: 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp. 548–550 (2017). https://doi.org/10.1109/EIConRus. 2017.7910613

[11] Deldjoo, Y., Schedl, M., Knees, P.: Content-driven music recommendation: Evolution, state of the art, and challenges. CoRR **abs/2107.11803** (2021) 2107.11803

[12] Van Meteren, R., Van Someren, M.: Using content-based filtering for recommendation. In: Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop, vol. 30, pp. 47–56 (2000). Barcelona

[13] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering **17**(6), 734–749 (2005) https://doi.org/10. 1109/TKDE.2005.99

[14] Zheng, Y.: Context-aware collaborative filtering using context similarity: An empirical comparison. Multidisciplinary Digital Publishing Institute (2022). https://doi.org/10.3390/info13010042

[15] Lye, G.X., Cheng, W.K., Tan, T.B., Hung, C.W., Chen, Y.-L.: Creating personalized recommendations in a smart community by performing user trajectory analysis through social internet of things deployment. Multidisciplinary Digital Publishing Institute (2020). https://doi.org/10.3390%2Fs20072098

[16] Bellman, R.E.: Adaptive Control Processes: A guided tour. Princeton University Press (2015)

[17] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science (New York, N.Y.) **313**, 504–7 (2006) https://doi.org/10.1126/science.1127647

[18] Pennington, J., Socher, R., Manning, C.: GloVe: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543. Association for Computational Linguistics, Doha, Qatar (2014). https://doi.org/10.3115/v1/D14-1162 . https://aclanthology.org/D14-1162

[19] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. CoRR **abs/1310.4546** (2013) 1310.4546

[20] Yu, R.: A Tutorial on VAEs: From Bayes' rule to lossless compression (2020). https://arxiv.org/abs/2006.10273

[21] Caruana, R.: Multitask learning - machine learning. Kluwer Academic Publishers (1997). https://link.springer.com/article/10.1023/A:1007379606734

[22] Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '02, pp. 253–260. Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/564376.564421

[23] Hoi, S.C.H., Sahoo, D., Lu, J., Zhao, P.: Online learning: A comprehensive survey. CoRR **abs/1802.02871** (2018) 1802.02871

[24] Shalev-Shwartz, S.: Online Learning and Online Convex Optimization, (2012). https://doi.org/10.1561/2200000018

[25] Kocur, J., Brady, S., Moseley, B., Shaffer, K., Langolf, L., Hanenberg, S., Lavengood, M.: Pitch and pitch class. Pressbooks (2021). https://viva.pressbooks.pub/openmusictheory/chapter/pitch-and-pitch-class/

[26] Web API Documentation. Spotify AB (2023). https://developer.spotify.com/ documentation/web-api

[27] Agarap, A.F.: Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375 (2018)

[28] Rumelhart, D., Hinton, G., Williams, R.: Learning representations by backpropagating errors. Nature (1986). https://doi.org/10.1038/323533a0

[29] Werbos, P., John, P.: Beyond regression : new tools for prediction and analysis in the behavioral sciences /. Harvard University (1974)

[30] Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recognit. **30**, 1145–1159 (1997)

[31] Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A.M., Jozefowicz, R., Bengio, S.: Generating sentences from a continuous space (2016). https://doi.org/10.48550/arXiv.1511.06349

[32] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders (2016). https://doi.org/10.48550/arXiv.1511.05644

[33] Pidhorskyi, S., Adjeroh, D., Doretto, G.: Adversarial latent autoencoders (2020). https://doi.org/10.48550/arXiv.2004.04467

[34] Nasios, N., Bors, A.G.: Variational learning for gaussian mixture models. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) **36**(4), 849–862 (2006) https://doi.org/10.1109/tsmcb.2006.872273