Genre Preference Trend Analysis Using Hidden Markov Model

William Lee WOOL@PRINCETON.EDU
Princeton University, Princeton, NJ 08544

Abstract

A temporal series of genre preferences of an individual is a very hard sequence to predict, as genre choices are heavily influenced by personal preferences, subjectivity, and other various factors such as mood. In this paper, we attempt to study trends of genre preferences of last.fm users using Hidden Markov Model, and make predictions of genres that a given user is likely to move on to. We train our model on a set of pairings (a_m, g_l), which are artist of current song, and genre of the next song, respectively. Our model performs nicely and makes high number of correct genre predictions, although it tends to overfit for users of smaller set of preferred genres.

1. Overview

1.1. Introduction

Surprisingly little has been written on the topic of automatic genre *definition*; indeed, virtually all papers on recommendation and discovery assume the presence of external genre meta-data. Work from the Music Information Retrieval (MIR) community almost invariably aims to match audio to a preexisting set of genre definitions rather than identify genre boundaries dynamically from user behavior. (The subjectivity of genre is usually mentioned in passing, often to explain performance limitations on classification algorithms; little in the way of genre redefinition results from these digressions.)

That most researchers skirt around the issue of genre is no surprise; debates about genre classification in music communities are virulent and ultimately unresolvable on account of their extreme subjectivity, and would require significant expertise in cultural analysis to begin solving.

The subjectivity of the genres naturally leads to modeling the dynamics of song choices of users using latent variables that describe users' genre preferences using probability models. Hidden Markov Model (HMM) is a great choice for modeling such system. HMM is comprised of hidden states $z_{1:T}$ and observed variables $x_{1:T}$; each observed variable is generated from a certain distribution with parameters unique to its corresponding hidden state, which is governed by a Markov chain system.

Two main types of HMM are supervised and unsupervised HMM. In unsupervised HMM, hidden states are estimated using Baum-Welch algorithm, which is a type expectation-maximization algorithm for training a HMM. This approach will not be covered in this paper, but it could provide more insight to the data as the model estimates the genres automatically using EM approach.

We will be training a supervised HMM on play histories of last.fm users. Each play record has information about the current song, and we use this to mine genre of the song, which will represent the hidden state of the model. Our goal is to use our trained HMM to make a prediction about a user's next choice of genre given the artist of the current song. This model can also be investigated to study the dynamics of last.fm users and temporal trends of their genre preferences.

All plots in this paper is created using `matplotlib` package in python language. (Hunter, 2007)

1.2. Dataset

Our primary dataset is the last.fm 1K dataset. (Celma, 2010) This dataset, after removing corrupt entries, contains the complete listening histories for 991 users, collected over a four-year period between 2005 and 2009. For each users, there are number of play records that have information about the current song played. We can extract artist and title of each song to retrieve the song's data using EchoNest or last.fm API, and create pairings (a_m, g_l) where a_m is the mth artist and g_l is the lth genre in the dataset. We use these pairings to train a HMM.

2. Related Work

2.1. Genre Classification

Our paper is focused on analyzing the genre preference trends of users and make predictions using trained HMM. However, HMM can also be applied to a set of songs to perform a classification. Unsupervised HMM uses Baum-Welch algorithm, which is a type of expectation-maximization algorithm, to find the optimal hidden state of each observed variable. In (Xi Shao & Kankanhalli, 2004), unsupervised HMM is trained using similarity metrics that are calculated from similarities in rhythmic structures of the songs. This approach is similar to ours in that it uses HMM to study the unobserved similarities that exist between different songs; nonetheless, while this method focuses on classifying songs based on their rhythmic similarities, our method learns trends of genres selected, and predicts genre of the next song. Because we model relationships between temporal genre sequence and a sequence of artists chosen, we are able to study the dynamics of users more closely.

Other work that can be used in our model to improve data labeling process is (Inoue & Urahama, 2001). In this work, a fuzzy clustering algorithm that accepts co-occurrence matrix as input and clusters data points is discussed. In our supervised model, decreasing the number of hidden states is a big problem, as discussed in section 4. We can apply this method to cluster genres into even smaller set of new labels that can be used as hidden states. We can collect a series of similar genres from music API to create a co-occurrence matrix of all the genres that we have, and classify each genre to a macro-group of similar genres. This will simplify our model by decreasing the number of parameters significantly. It is also suitable in clustering genres, because a genre is similar to many different genres on different level of similarity. However, selecting the proper number of classes is another optimization problem that we face.

2.2. Genre Prediction

There are also papers that researched predicting genre, or "social tags", of a song based on its acoustic information. (Eck et al., 2008) and (Bertin-Mahieux et al., 2010) uses boosting methods to make predictions of genre tags of a song using the mp3 data of the song. For each song s_i and a tag t_j, multiple classifiers are used to make predictions whether the song has: no features of tag t_j, some features of tag t_j, and a lot of features of tag t_j. Then, based on the distribution of classes predicted and weights of each weak classifier, an appropriate tag is predicted. This is another approach that we can try in achieving a better set of labels for labeling hidden states of the model.

3. Preliminary Data Analysis

Before approaching the dataset with HMM technique, we used simple statistical tests and clustering methods to explore the dataset and its limitations.

3.1. Distribution

A preliminary investigation revealed that, like natural language, the data are power-law distributed:

$$p(x) \propto x^{-\alpha}$$

Power law fit was calculated using the Kolmogorov-Smirnov test, selecting the minimum rank x for which power law scaling holds by repeated application of the KS test to minimize the distance between the data and the fitted power law.

The best-fit exponents α, in contrast to those of natural language vocabularies, where $alpha \approx 1$, are quite large, indicating a steep fall-off for unpopular songs (Piantadosi, 2014).

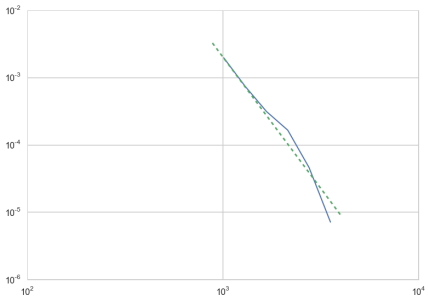

Figure 1. Best-fit power law plot for song frequencies. Estimated exponent: 3.90. Minimum rank for power law fit: 885 (out of 1,498,714 songs).

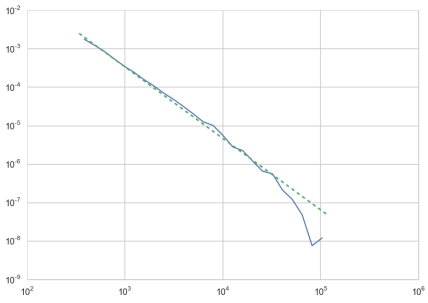

Figure 2. Best-fit power law plot for artist frequencies. Estimated exponent: 1.86. Minimum rank for power law fit: 340 (out of 173,919 artists).

3.2. PCA

First, we created a sparse matrix M, where M_{ij} represents the number of times user i played song j. This matrix has 1,486,727,264 entries, 0.3% of which were nonzero. Using this matrix, we hoped to cluster the songs and users into genres and 'user personality types' respectively. Following the example of the Netflix problem, we assumed low rank, which corresponds to the existence of a few archetypal users (users which everyone else is a combination of), where an archetypal user is represented by a playlist.

Our goal in this exploratory analysis was to see if the play matrix carried enough information to cluster songs in some reasonable way. Since the play matrix does not understand genre, or use the artist directly as a feature, we decided on PCA as a method to recover genre information in the play history of users.

We could not center the data, because doing so would destroy sparsity. Therefore, we expected the first principal component to be uninformative of cluster assignments, capturing the mean of the data instead.

We chose to use the log of the number of plays instead of the number of plays itself. This had two advantages: reducing the effect of outliers, and providing strong differentiation between the separate situations of no plays, some plays, and many plays (i.e. unfamiliar, familiar (possibly hinting at genre similarity), and 'favorite song'). As an added convenience, it removed songs that were played only once, which are problematic to weight heavily, given their overwhelming presence in the dataset. For illustration, here are the top entries of the first and second component (after the first junk component). We can see that PCA worked well, clustering songs that are related (as judged by our intuitive mental models of musical genre).

Component Number: 1

29242 Metallica: Nothing Else Matters	0.039512583237
40550 Metallica: Master Of Puppets	0.0390164041261
40552 Metallica: One	0.0368443433159
32911 Metallica: Enter Sandman	0.0367895312332
10941 Dread Zeppelin: Stairway To Heaven	0.0365366856085
33470 Metallica: Sad But True	0.0329327752888
33455 Metallica: The Unforgiven	0.0327713739416
37227 Metallica: Fade To Black	0.0325640601586
12321 Guns N' Roses: Welcome To The Jungle	0.0315305562435
38146 Metallica: Battery	0.0314776978129
63811 Queen: Bohemian Rhapsody	0.0301365930856
38119 Metallica: Wherever I May Roam	0.0301281480811
18800 Guns N' Roses: November Rain	0.0301080962211
12132 Black Sabbath: Paranoid	0.0295927419139
38117 Metallica: For Whom The Bell Tolls	0.0294124525187
32969 System Of A Down: B.Y.O.B.	0.0287993789382
42995 Motrhead: Ace Of Spades	0.0285907857447
40569 Guns N' Roses: Paradise City	0.0285214956135
32244 Pink Floyd: Comfortably Numb	0.0284861119442
61775 Guns N' Roses: Sweet Child O' Mine	0.0281871832982

Component Number: 2

6100 Snow Patrol: Chasing Cars	0.0379937354037
3238 Snow Patrol: Run	0.0351394834574
5976 Keane: Somewhere Only We Know	0.0346835391398
5697 Coldplay: Fix You	0.0340572087773
9269 The Killers: When You Were Young	0.03351177052
4690 The Fray: How To Save A Life	0.0329460616405
633 Coldplay: Viva La Vida	0.0321872938911
5694 Coldplay: Speed Of Sound	0.0314566624118
7185 Snow Patrol: Chocolate	0.0314232865266
7862 The Killers: Read My Mind	0.0311445037731
6102 Snow Patrol: You'Re All I Have	0.030293996609
9265 The Killers: Bones	0.0301016741267
2163 Mika: Grace Kelly	0.0300161921033
6093 Snow Patrol: Open Your Eyes	0.0294973318651
6245 Coldplay: Clocks	0.0293846185055
5120 Coldplay: The Scientist	0.0285674019869
5052 The Kooks: Nave	0.0277590241249
8447 Franz Ferdinand: Take Me Out	0.0277189567374
28975 Maroon 5: She Will Be Loved	0.0269499599944
6088 Snow Patrol: How To Be Dead	0.0268015040182

This validates our supposition that there are some 'archetypal' users represented by genres of songs.

3.3. Bigram clustering

For this method of clustering, we posited that the next song in each user's sequence is chosen in a purely Markovian way, i.e. the next song depends only on the current song being played. This is a gross oversimplification (with the advantage of being extremely fast to run), making it a perfect exploratory probing of temporal dependency.

We trained bigram models by running through the play data, either for a given user or for the whole dataset at once. More precisely, for every song, we found the empirical distrubition of the next song by running through the data and counting. This resulted in a dictionary from each song s to its associated next-song probability distribution π_s. We put two songs i, j in the same cluster if either i was the argmax of π_j, or j was the argmax of π_i. We then sorted each cluster by playcount, so that we had a well-defined ordering. This worked extremely well. To illustrate, here are the first two clusters when we look at user 3:

CLUSTER 0
The Killers: When You Were Young
The Killers: Read My Mind
The Killers: Bones
The Killers: For Reasons Unknown
The Killers: Sam'S Town
Queen: We Will Rock You
Bloc Party: The Prayer
Bloc Party: I Still Remember
The Killers: Bling (Confession Of A King)
The Killers: This River Is Wild
The Killers: Enterlude
Michael Jackson: Smooth Criminal
The Black Eyed Peas: My Humps
The Killers: Uncle Jonny
The Killers: Exitlude
The Killers: Where The White Boys Dance
The Killers: All The Pretty Faces
Bloc Party: Uniform
Bloc Party: Waiting For The 7.18
The Killers: My List

CLUSTER 1
Peter Bjorn And John: Young Folks
Rihanna: Umbrella (Feat. Jay-Z)
Burial: Archangel
Groove Armada: My Friend
Mylo: Paris Four Hundred
Rammstein: Mein Herz Brennt
Burial: [Untitled]
Burial: Ghost Hardware
Burial: Etched Headplate
Burial: In Mcdonalds
Burial: Near Dark
Burial: Endorphin
Red Hot Chili Peppers: 21St Century
Burial: Untrue
Klaxons: Four Horsemen Of 2012
Burial: Shell Of Light
Burial: Dog Shelter
Burial: Raver
Burial: Homeless
Burial: Uk

This worked far too well: the result aligns with the assumption that listening habits are best modeled under the assumption that listeners choose an artist or an album and play through their songs sequentially. While this may be true, it is not particularly illustrative of genre, and we may wish to disregard play order to avoid this pattern dominating the analysis. This gives further justification for a *bag-of-songs* approach.

3.4. HMM Clustering

We then tried a more sophisticated Markovian clustering algorithm. To avoid the overfitting of the bigram model, we instead modeled play history by assuming that the underlyring genres of songs is a Markov chain. We trained user-by-user HMMs (it was no longer tractable to do the whole dataset) with ten components. Then, for each component, we printed out the twenty songs with the highest emission probability. Below are the first two components for User 5 (notice that the first three components of bigram clustering and HMM clustering are not likely to overlap, since the ordering of components has no reason to be the same). The hope is that the algorithm clusters all of the similar song (same artist/album), and gives insight on how the users switch between genres. If user tendencies or underlying recommendation algorithms make it so each session is of one genre type, which seems to be the case, HMM clustering may split by play session.

COMPONENT 0
Ellen Allien & Apparat: Way Out
Ellen Allien & Apparat: Turbo Dreams (Pier Bucci Remix)
Ellen Allien & Apparat: Jet
The Postal Service: The District Sleeps Alone Tonight
Fischerspooner: Get Confused
Ellen Allien & Apparat: Retina
Ellen Allien & Apparat: Leave Me Alone
Modjo: Lady
Ricardo Villalobos: Ichso
Ellen Allien & Apparat: Under
Ellen Allien & Apparat: Edison
Ellen Allien & Apparat: Floating Points
Luca Bacchetti: What Ever
Lawrence: Leave Me Tomorrow
Fischerspooner: Never Win
Sakura: 15. S.M.S.
Ellen Allien & Apparat: Metric
Luca Bacchetti: Flat Mates
Ellen Allien & Apparat: Turbo Dreams
Lawrence: Crippled Trees

COMPONENT 1
Placebo: Meds
Placebo: Red
Placebo: Post Blue
Placebo: Space Monkey
Placebo: Follow The Cops Back Home
Placebo: Drag
Ryksopp: What Else Is There
Oxia: Not Sure
Trentemller: Moan (Trentemller Remix)
Nathan Fake: Outhouse (Main Mix)
Extrawelt: Doch Doch
Swayzak: Smile And Receive
Babyshambles: Fuck Forever
Gui Boratto: Like You (Supermayer Mix)
Swayzak: Quiet Life
Apparat: Error404
Swayzak: So Cheap
:
Apparat: Multifocus
Nathan Fake: Outhouse (Beaty Tool Mix)

We observed similar results for HMM clustering as for bigram clustering: clustering was performed mostly by artist. The components are slightly more mixed in this case, but it is not clear if this is capturing some additional structure, or if the model is simply performing worse at accomplishing what bigram clustering accomplishes. The added complexity of the HMM should avoid the problem of overweighting albums and artists suffered by the naive bigram model.

4. Genre prediction using Hidden Markov Model

4.1. Rationale

Hidden Markov Model (HMM) is able to capture the probabilistic Markovian structure among the latent variables of the model that governs the distributions observed variables, allowing us to investigate dynamics of a system over time. We aim to make a prediction of genre of the song played based on artist of the current song that the user is listening to. HMM that is trained on last.fm data, in which records of plays of users are stored, is able to make such inference. We shall assume that genre of the next song is hidden state, and that the artist of the current song is generated from a distribution unique to that genre. This way, we are able to model dynamics of genres (hidden state) and artists (observed variable) simultaneously but separately, and use inference tools such as filtering and smoothing process to study the characteristics of the genre preferences of users on last fm.

As seen in section 3, we caught some patterns in HMM-clustered results that hinted at potential Markovian structure among the songs. Therefore, HMM could return promising results when applied to this data set. The crucial preprocessing procedure for this analysis is genre labeling for each song. This will determine the latent variables of the model. Different methods of collecting genre information have been tried using EchoNest API and last.fm API, which is discussed in section 4.3.

4.2. Model Description

The joint distribution of a HMM model is

$$p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}) = p(\mathbf{z}_{1:T})p(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) = \left[p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\right]\left[\prod_{t=1}^{T} p(\mathbf{x}_t|z_t)\right]$$

We can see that from Bayes rule, the joint distribution is divided into joint distributions of hidden state and observed state. Therefore dynamics of Hidden Markov Model is governed by combination of the Markov chain properties of the hidden states $z_{1:T}$ and the distribution of observed states $x_{1:T}$ given hidden states $z_{1:T}$, which is unique for each hidden state z_t. This is shown in Figure 3. In our model we assume that given a genre (hidden state, or latent variable), the probability distribution of the artists (observed variable) is categorical distribution unique to the hidden state. Upon fitting parameters of the model, we can evaluate how well our model describes the last.fm data and study the dynamics of user genre preferences using HMM inference techniques such as filtering, smoothing, and Viterbi algorithms.

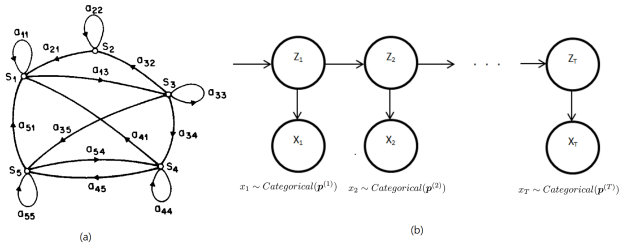

Figure 3. (a) Markovian dynamics of hidden states. Each state S_i represents genres of next song g_l. Transition probabilities are $a_{ij} = A(i, j)$, which is fitted during training process. (b) Hidden Markov Model dynamics. Hidden states z_t, which takes values from set of genres $\{g_l\}_{1:L}$ transit from one state to another in Markovian fashion, determined by the transition matrix $\boldsymbol{A}$. Each observed variable x_t, which takes values from set of artists $\{a_m\}_{1:M}$, is generated from a categorical distribution unique to each hidden state.

4.3. Data

4.3.1. Genre Tagging

We trained this model on (song, next-genre) pairings extracted from last.fm data. This model was fully supervised, as we had labels for *almost* every song, retrieved from EchoNest and last.fm API. There were two major problems in genre data collected: the number genre tags returned by APIs were too large to quickly train a nicely fit HMM, and some songs did not have any genre information.

Both EchoNest and last.fm returned genre set that is very large, causing the model to have excessively many hidden states, which makes computation speed and memory consumption much slower and higher. EchoNest API returned 772 genres for 65,020 artists when trained on the full data. Last.fm API returns tags of a song added by users, rather than a specific genre, causing it to have much more data. The API itself also seemed unstable, as it was not working until December, and some of its methods still do not work. When applied on only 200 users, which was roughly 52,000 artists, last.fm API already had returned around 18,000 genres.

Such large number of genres means that the parameters of the model, which is discussed in next section, are very large and sparse. This causes problems of slow computation and large memory consumption. One natural way of solving this problem is classifying or clustering genres into a set of fewer macro-genres that are groups of genres. There are many different classification or clustering methods that can be used to decrease the number of genres; many are also discussed in section 3.

We aim to classify or cluster genres based on the genre's similarity to other genres, and thankfully, EchoNest and last.fm have methods that return similar genres of a given genre. However, method in last.fm API does not work currently. Therefore EchoNest API, which returns a list of similar genres with similarity scores for a given genre, can be used to create a similarity matrix and classify or cluster genres using this matrix; however, we were not able to try out this method yet due to time constraints.

Other major problem of processing dataset is labeling songs that do not have any genre data on EchoNest or last.fm. This problem can be addressed by estimating the genres as best we can using clustering, binning, brute-force, etc. First method that was considered was to simply label all unidentifiable songs as *unknown*. However, this would disrupt the Markovian dynamics of the hidden states. During training phase, the *unknown* state will be encountered close to many different types of genres, and this could affect fitting process of the Markov transition probability matrix. Another way is to simply use EM algorithm to fit an unsupervised HMM, then use the top genre labels to estimate a good starting point for Baum-Welch algorithm. In this paper we simply label them as *unknown*; the number of songs whose data that wasn't retrieved was infinitesimal compared to the total number of songs.

From here on, we will only discuss data processed using EchoNest API. The distributions of artists and genres retrieved using EchoNest follow power rule, as seen in exploratory analysis of our data. Figure 4 shows plots of frequencies of artists and genres. $\alpha_a = 1.83$ for artists, and $\alpha_g = 1.96$ for genres. This indicates that a small top percentage of the artists and genres of the data set is

played mostly, and there are many artists and genres that are played very few times.

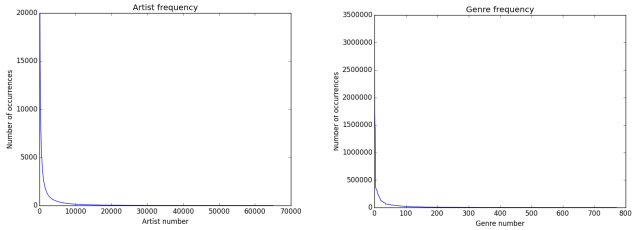

Figure 4. Distributions of artists and genres retrieved using EchoNest. $\alpha_a = 1.83$, and $\alpha_g = 1.96$, where $p(x_) \propto x_i^{\alpha_i}$ for $i \in (a, g)$

4.3.2. LAST.FM 1K DATASET

In aggregate, the last.fm 1K dataset is a .tsv file of 19,131,673 rows, an average of 19,305 songs per listener. From each row, we extract userid, timestamp, artist name, and track name. First, rows are grouped into each user. Then, using timestamps, each user's rows are sorted chronologically. Using the artist name and track name, genre of each song is retrieved using EchoNest. If the song is not found, we then use the given artist to get the top genre of the artist. (This could pose problems, as some artists have few types of genres and this process could return non-optimal genre.) If even the artist is not found, we simply label the song's genre as *unknown*. Finally, each user data is stored as a $T_i \times 2$ matrix, where T_i is the number of pairs of artist and genre for user i. Each row is a pair (a_t, g_t), which are artist of currently played song at time t and genre of the next song played at time $t+1$, with $a_t \in \{a_m\}_{1:M}$ and $g_t \in \{g_l\}_{1:L}\ \forall t \in 1, \cdots, T_i$. (Each artist is from a set of 65,020 artists, and each genre is from a set of 772 genres) Therefore T_i is 1 less than the total number of plays, and genre of the first play record and artist of the last play record were not used. There was a user whose play count was 2, reducing the processed play history to length of one; therefore it was discarded.

These user matrices are ready to be trained. Each a_t is an observed variable x_t, and each g_t is a hidden variable z_t, for each user. z_t moves onto next hidden state z_{t+1} based on a Markovian transition probability matrix, and each x_t is drawn from a categorical distribution with probability parameters unique to its hidden state; that is, each genre has unique probability parameter $\boldsymbol{p^l}$, and $p(x_t = a_m, |z_t = g_l) = p^l(m), m \in 1, \cdots, M$.

Another problem with the dataset is its large number of artists. It naturally leads to problem caused by artists in test set has not been seen during training phase. This causes problem in inference process while incorporating the class-conditional distributions of genres for that artist, $p(x_t|z_t = g_l)$, as the model has no information about the artist. In this paper, we get past this problem by simply setting $p(x_t|z_t = g_l) = 1$. This allows the model to think that this artist is very likely to be observed variable for every genre. This needs to be checked if it breaks the mathematical validity of the model. Other method of solving this problem is to lower the number of artists by another method, so that we are working with a group of artists rather than an artist as our observed variable.

4.4. Training

The parameters to be estimated are $\theta = (\boldsymbol{\pi}, \mathbf{A}, \mathbf{B})$: $\boldsymbol{\pi}_i = \pi(i) = p(z_1 = g_i)$, $\boldsymbol{A}_{ij} = A(i,j) = p(z_{t+1} = g_j|z_t = g_i)$, and $\boldsymbol{B}_{jk} = B(j,m) = p(x_t = a_m|z_t = g_j)$. $\boldsymbol{\pi}$ is the initial latent probability distribution of length $L = 772$. $\boldsymbol{A}$ is a $L \times L$ Markovian transition probability matrix. $\boldsymbol{B}$ is a $L \times M$ matrix, whose row is a categorical distribution of artists for genre g_l, which is of length $M = 65020$. $\boldsymbol{\pi}$ and $\boldsymbol{A}$ model hidden state transition dynamics, while $\boldsymbol{B}$ contains information about distributions of observed states, given a hidden state. HMM used in this paper was coded in python by referencing (Murphy, 2012) and (Rabiner)

The number of parameter entries to be estimated grows exponentially in number of genres, L, and number of artists, M. With EchoNest-labeled data, we need to estimate a total of $L + LM + L^2 \approx 50$M parameters. With large L and M that we are dealing with, parameter training takes very long time and lots of memory.

We can fit the parameters using simple maximum likelihood approach, as we have genre labels for each song. $\boldsymbol{\pi}$ and $\boldsymbol{A}$ are calculated using MLE approach for a Markov Model:

$$\pi(j) = \frac{N_j^1}{\sum_j N_j^1}, A(j,k) = \frac{N_{j,k}}{\sum_k N_{j,k}}$$

which are normalized count of

$$N_j^1 = \sum_{i=1}^{N} \mathbf{1}(x_{i,1} = a_j)$$
$$N_{j,k} = \sum_{i=1}^{N} \sum_{t=1}^{T_i - 1} \mathbf{1}(x_{i,t} = a_j, x_{i,t+1} = a_k)$$

where T_i is the number of artist-genre pairings in matrix of user i. Also, x_t and z_t of user i are denoted as $x_{i,t}$ and $z_{i,t}$ in this section of the paper.

In our model, in which we assume x_t to have categorical distribution over artists $a_1, \cdots, a_M$ given a hidden state z_t, the MLE estimate of $\boldsymbol{B}$ is:

$$B(j,m) = \frac{N_{j,m}^X}{N_j}$$

which is normalized count of

$$N^X_{j,m} = \sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathbf{1}(z_{i,t} = g_j, x_{i,t} = a_m)$$
$$N_j = \sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathbf{1}(z_{i,t} = g_j)$$

4.5. Inference

When the parameters are estimated using training set, we can perform forwards, backwards, and a mixture of the two algorithms to make different kinds of prediction. (Murphy, 2012) Filtering, smoothing, and MAP estimation methods are discussed below.

Filtering and smoothing use given parameters to iteratively calculate, for each t, the probability of being in each hidden state, given a set of observed variables, $x_{1:t}$ and $x_{1:T}$, respectively, using MLE approach. MAP estimation uses Viterbi algorithm, which is a type of EM algorithm, to estimate the most likely sequence of hidden states.

4.5.1. Filtering

Filtering in HMM inference literature indicates computing belief state $p(z_t|x_{1:t})$ online, using information collected up to time t. Taking previous observed states into account reduces noise in the predicted values than just simply estimating $p(z_t|x_t)$.

The filtered marginal can be expressed, using Bayes rule, as

$$\begin{aligned}
\alpha_t(j) &= p(z_t = g_j|x_{1:t}) \\
&= p(z_t = g_j|x_t, x_{1:t-1}) \\
&\propto p(x_t|z_t = g_j, x_{1:t-1})p(z_t = g_j|x_{1:t-1}) \\
&= p(x_t|z_t = g_j) \\
&\quad \times \sum_{l=1}^{L}[p(z_t = g_j|z_{t-1} = g_l)p(z_{t-1} = g_l|x_{1:t-1})] \\
&= \psi_t(j)\sum_{l=1}^{L}[\Psi(l,j)\alpha_{t-1}(l)] \\
&\Rightarrow \\
\boldsymbol{\alpha}_t &\propto \boldsymbol{\psi}_t \odot (\boldsymbol{\Psi}^T \boldsymbol{\alpha}_{t-1})
\end{aligned}$$

where $\boldsymbol{\alpha_t}$ is a vector whose entries are $\alpha_t(j)$, $\boldsymbol{\psi}_t$ is a vector whose entries are $\psi_t(j) = p(x_t|z_t = g_j)$, $\boldsymbol{\Psi}$ is a matrix whose entries are $\Psi(i,j) = p(z_t = g_j|z_{t-1} = g_i)$, and $\odot$ is Hadamard product, which is an element-wise vector multiplication. Notice that $\boldsymbol{\psi}_t$ is the column m of parameter $\boldsymbol{B}$, where $a_m = a_t$. Also, $\boldsymbol{\Psi} = \boldsymbol{A}$. We also use $\boldsymbol{\pi}$ in replacement for $\boldsymbol{\alpha}_0$. Because the result of multiplication of parameters is not equal to but only proportional to the alpha values, we need to normalize the result at every iteration t to get a probability distribution in which the values sum up to 1. Pseudocode of the filtering algorithm is provided below.

⟨Forwards algorithm⟩
Require: $\boldsymbol{\pi}$, $\boldsymbol{\Psi}$, $\{\boldsymbol{\psi}_t\}_{1:T}$
  $\boldsymbol{\alpha}_1 = \text{normalize}(\boldsymbol{\pi} \odot \boldsymbol{\psi}_1)$
  for $t \in (2, \cdots, T)$ **do**
    $\boldsymbol{\alpha}_t = \text{normalize}(\boldsymbol{\psi}_t \odot (\boldsymbol{\Psi}^T \boldsymbol{\alpha}_{t-1}))$
  end for
  return $\boldsymbol{\alpha}_{1:T}$

Using these filtered marginals, we can make predictions on the most likely genre of the next song given a play history up to time t. Evaluation of this inference method will be shown in section 4.5.4.

4.5.2. Smoothing

Smoothing in HMM aims to calculate the smoothed marginal $p(z_t|x_{1:T})$, which uses information collected from time 1 to T. This is offline counterpart of the filtering process. In smoothing, we also take future data into account when calculating probability of a hidden state, and this reduces uncertainty significantly.

For smoothing, we calculate $\gamma_t(j) = p(z_t = g_j|x_{1:T})$ using $(\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_T)$ and $(\boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_T)$ where $\boldsymbol{\alpha}_t$ is the filtered marginals discussed in section 4.5.1, and $\beta_t(j) = p(x_{t+1:T}|z_t = g_j)$. Using more Bayes rule, we get

$$\begin{aligned}
\beta_{t-1}(j) &= p(x_{t:T}|z_{t-1} = g_j) \\
&= \sum_i p(z_t = g_i, x_t, x_{t+1:T}|z_{t-1} = g_j) \\
&= \sum_i p(x_{t+1:T}|z_t = g_i, x_t, z_{t-1} = g_j) \\
&\qquad \times p(z_t = g_i, x_t|z_{t-1} = g_j) \\
&= \sum_i p(x_{t+1:T}|z_t = g_i)p(z_t = g_i, x_t|z_{t-1} = g_j) \\
&= \sum_i p(x_{t+1:T}|z_t = g_i)p(x_t|z_t = g_i, z_{t-1} = g_j) \\
&\qquad \times p(z_t = g_i|z_{t-1} = g_j) \\
&= \sum_i p(x_{t+1:T}|z_t = g_i)p(x_t|z_t = g_i) \\
&\qquad \times p(z_t = g_i|z_{t-1} = g_j) \\
&= \sum_i \beta_t(i)\psi_t(i)\Psi(j,i) \\
&\Rightarrow \\
\boldsymbol{\beta}_{t-1} &= \boldsymbol{\Psi}(\boldsymbol{\psi}_t \odot \boldsymbol{\beta}_t)
\end{aligned}$$

The base case is

$$\beta_T(j) = p(x_{T+1:T}|z_T = g_j) = p(\emptyset) = 1$$

Starting with this base case, we iteratively calculate $\boldsymbol{\beta}_t$ for all t from $T-1$ to 1.

Pseudocode of backwards algorithm for calculating $\{\boldsymbol{\beta}_t\}_{1:T}$ is provided below. Note that unlike $\boldsymbol{\alpha}_t$, we do not necessarily have to normalize $\boldsymbol{\beta}_t$ as it is not a probability distribution; however, we still normalize it at every iteration to prevent numerical underflow.

⟨Backwards algorithm⟩
Require: $\boldsymbol{\Psi}$, $\{\boldsymbol{\psi}_t\}_{1:T}$
  $\boldsymbol{\beta}_T = (1, \cdots, 1)$
  for $t \in (T-1, \cdots, 1)$ **do**
    $\boldsymbol{\beta}_t = \text{normalize}(\boldsymbol{\Psi}(\boldsymbol{\psi}_{t+1} \odot \boldsymbol{\beta}_{t+1}))$
  end for
  return $\boldsymbol{\beta}_{1:T}$

After $\{\boldsymbol{\alpha}_t\}_{1:T}$ and $\{\boldsymbol{\beta}_t\}_{1:T}$ have been calculated, we multiply and normalize them to get smoothed marginals:

$$\begin{aligned}\gamma_t(j) &= p(z_t = g_j|x_{1:T}) \\ &\propto p(z_t = g_j, x_{t+1:T}|x_{1:t}) \\ &\propto p(z_t = g_j|x_{1:t})p(x_{t+1:T}|z_t = g_j, x_{1:t}) \\ &= p(z_t = g_j|x_{1:t})p(x_{t+1:T}|z_t = g_j) \\ &= \alpha_t(j)\beta_t(j) \\ &\Rightarrow \\ \boldsymbol{\gamma}_t &\propto \boldsymbol{\alpha}_t \boldsymbol{\beta}_t\end{aligned}$$

Pseudocode for forwards-backwards algorithm is provided below.

⟨Forwards - backwards algorithm⟩
Require: $\boldsymbol{\pi}$, $\boldsymbol{\Psi}$, $\{\boldsymbol{\psi}_t\}_{1:T}$
  Calculate $\boldsymbol{\alpha}_{1:T}$ using forwards algorithm
  Calculate $\boldsymbol{\beta}_{1:T}$ using backwards algorithm
  for $t \in (1, \cdots, T)$ **do**
    $\boldsymbol{\gamma}_t = \text{normalize}(\boldsymbol{\alpha_t} \cdot \boldsymbol{\beta}_t)$
  end for
  return $\boldsymbol{\gamma}_{1:T}$

Similar to filtering inference, we can use these probabilities to make prediction on the genre at time t given the full play history of the user in offline mode.

4.5.3. MAP ESTIMATION

Unlike filtering and smoothing inference in which we use maximum likelihood estimation to fit parameters of the model, MAP estimation using Viterbi algorithm calculates the most likely hidden state sequence.

We aim to compute

$$\boldsymbol{z}^* = (z_1^*, \cdots, z_T^*) = \arg\max_{z_{1:T}} p(z_{1:T}|x_{1:T})$$

This is same as finding a shortest path through the trellis diagram, shown in Figure 5. Each node represents a possible hidden state, and each path is weighted according to transition and class-conditional probabilities. For a given path $z_{1:T}$ and its corresponding genre index set $i_{1:T}$, where $z_t = g_{i_t}$ for all t, the total weight is

$$\log \pi(i_1) + \log \psi_1(i_1) + \sum_{t=2}^{T} [\log \Psi(i_{t-1}, i_t) + \log \psi_t(i_t)]$$

which is the log likelihood of the path taken.

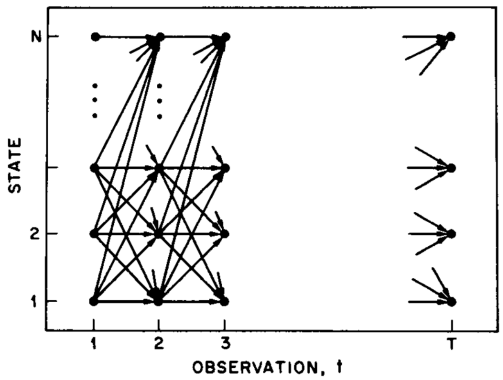

Figure 5. Trellis diagram representing HMM. Each node is a possible hidden state (genre), and each path is weighted, according to transition probabilities and class-conditional densities. Viterbi algorithm looks for path that has shortest path, or smallest total weights.

The shortest path $\boldsymbol{z}^*$ is computed by using max-product in a forward pass and using traceback in a backward pass. Let $\delta_t(j) = \max_{z_{1:t-1}} p(z_{1:t-1}, z_t = g_j|x_{1:t})$, which is the probability of ending up in genre j at time t, given that the previous path was the most probable path. We can rewrite the above in a recursive fashion, because the most probable path to g_j at t must consist of the most probable path to some genre g_i at $t-1$, followed by a transition from g_i to g_j. Thus $\delta_t(j) = \max_i \delta_{t-1}(i)\Psi(i,j)\psi_t(j)$. We also define and keep track of the index of the most likely previous genre: $\mathcal{P}_t(j) = \text{argmax}_i \delta_{t-1}(i)\Psi(i,j)\psi_t(j)$.

The algorithm is outlined below. It is divided into a forward pass and a backward pass; in forward pass, we calculate δ_t and $\mathcal{P}_t$ for each t. After calculating δ_t for all $t \in 1, \cdots, T$, we compute the index of most probable final state i_T, which is the index that returns the highest $\delta_T(\cdot)$.

Then we do a backwards pass and compute the most probable sequence. When all $\{i_t\}_{1:T}$ are calculated, we convert it to $\{z_t\}_{1:T}$ and return it.

⟨Viterbi algorithm⟩
Require: $\boldsymbol{\pi}$, $\boldsymbol{\Psi}$, $\{\boldsymbol{\psi}_t\}_{1:T}$
  $\delta_1 = \boldsymbol{\pi} \cdot \boldsymbol{\psi_1}$
  for $t \in (2, \cdots, T)$ **do**
    for $j \in (1, \cdots, L)$ **do**
      $\delta_t(j) = \max_i \delta_{t-1}(i)\Psi(i,j)\psi_t(j)$
      $\mathcal{P}_t(j) = \text{argmax}_i \delta_{t-1}(i)\Psi(i,j)\psi_t(j)$
    end for
  end for
  $i_T = \text{argmax}_i \delta_T(i)$
  $z_T = g_{i_T}$
  for $t \in (T-1, \cdots, 1)$ **do**
    $i_t = \mathcal{P}_{t+1}(i_{t+1})$
    $z_t = g_{i_t}$
  end for
  return $z_{1:T}$

To prevent numerical underflow from happening, we work in log-domains. This is possible, unlike in filtering or smoothing, because log max = max log, whereas log $\sum \neq \sum$ log.

This inference method has been coded, but we were unable to debug errors that are returning nonsense results; the training time is also very slow with such immense data set. Further work will be done on resolving this issue so that Viterbi algorithm can be compared to filtering and smoothing algorithms.

4.5.4. Results

With probabilities of each genre from filtering and smoothing inference, we can evaluate the fit of the model. For a given user, true genre of song at time t, z_t, is compared to lists of predicted genres $\{z^f_{t,i}\}_{i\in 1:h}$ and $\{z^s_{t,i}\}_{i\in 1:h}$, which are lists of genres with ith highest probability for filtering and smoothing, respectively, and h is a threshold of top number of genres. We consider $h = 5, 10$ for each method, and see if any of the top predictions matches the true genre at time t for a test set. Prediction accuracy for each model is calculated as percentage of matches, where a match occurs if $z_t \in \{z^{model}_{t,i}\}_{i\in 1:h}$ for $model \in \{f, s\}$. Because of immense time and memory complexity of HMM combined with big data such as last.fm data, we were only able to train the first 400 users as our training set. 100 users were used as our testing set.

Figure 6 shows boxplots of each models. We see that as h is increased, prediction accuracy values converge around a higher mean more tightly. However, when h is increased to 10, fourteen data points, most of which had prediction accuracy of almost zero, become outliers. We see that smoothing outperforms filtering in both cases.

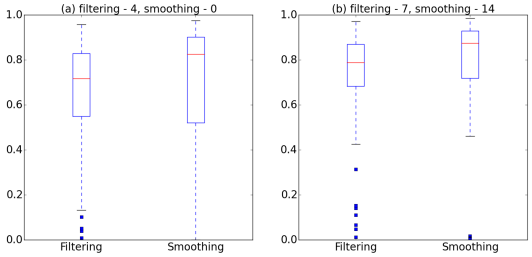

Figure 6. Boxplots of prediction accuracy with (a) $h = 5$ and (b) $h = 10$, for filtering and smoothing.

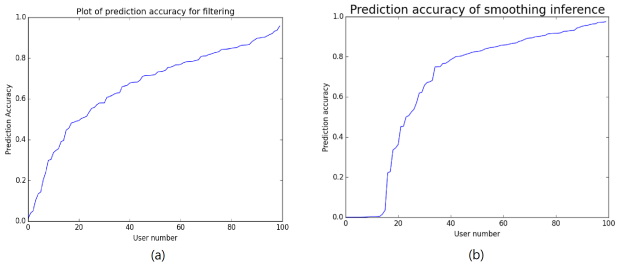

Figure 7. Plots of prediction accuracy for each user for filtering and smoothing inference with $h = 5$.

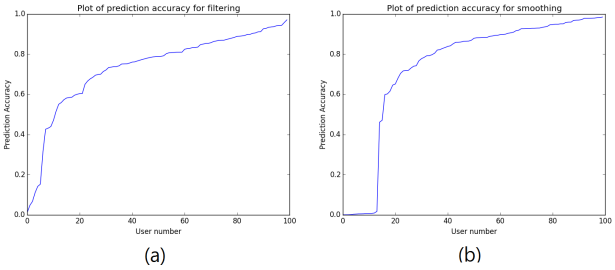

Figure 8. Plots of prediction accuracy for each user for filtering and smoothing inference with $h = 10$.

Figure 7 and 8 are plots of prediction accuracy of each user, for filtering and smoothing inference, with threshold set as 5 and 10. We see that filtering and smoothing return different distributions. While filtering manages to achieve prediction accuracy values that are significantly higher than for every user, smoothing has prediction value of almost zero for around $\sim 15\%$ of the users, which seems to be a boundary for smoothing algorithm that cannot be overcome even when h increases. This seems to be the 14 outliers seen in Figure 6(b) in smoothing case. However, as seen in Table 1 - 4, smoothing outperforms filtering in both mean and median of the prediction accuracy.

Distributions of smoothed marginals exhibit interesting groupings of users; consider figure 7(b) and 8(b). We see that the users are divided into three big groups. There is a small group of people whose prediction accuracy of the model is almost zero, who are likely to have ave genre preference trends that are different from what has been seen during the training phase. There is also a small group of people whose accuracy curve is very steep, who are likely to have genre preference trends that are mixes of what has seen before and not. There is a bigger group whose accuracy values are pretty high and the learning curve is relatively gradual whose genre trends are explained nicely by our model.

Table 1. Test results of filtering method with $h = 5$

Mean accuracy $= 0.658$	
Median accuracy $= 0.718$	
g_l with highest hit frequency	rock, metal, indie, pop, christmas, synthpop, polish, hip, jazz, house
g_l with highest miss frequency	rock, reggae, metal, pop, hip, jazz, indie, christmas, electronic, wave
a_m with highest hit frequency	The Knife, Nine Inch Nails, The 69 Eyes, The Beatles, Radiohead, Metallica, Pink Floyd, Coldplay, Evanescence, Frederic Chopin
a_m with highest miss frequency	name$_1$, name$_2$, M-Flo, Soda Stereo, Bonnie Pink, Bob Marley, L'Arc En Ciel, Los Fabulosos Cadillacs, name$_3$, Nine Inch Nails
Average predict index $= 1.780$	

Table 2. Test results of smoothing method with $h = 5$

Mean accuracy $= 0.668$	
Median accuracy $= 0.825$	
g_l with highest hit frequency	rock, metal, indie, pop, christmas, synthpop, polish, industrial, electronic, wave
g_l with highest miss frequency	rock, reggae, metal, pop, hip, indie, jazz, classical, house, japanese
a_m with highest hit frequency	The Knife, Nine Inch Nails, The 69 Eyes, The Beatles, Metallica, Radiohead, Pink Floyd, Coldplay, Evanescence, Pearl Jam
a_m with highest miss frequency	name$_1$, name$_2$, M-Flo, Frdric Chopin, Soda Stereo, Bonnie Pink, L'Arc En Ciel, Bob Marley, Tenhi, Los Fabulosos Cadillacs
Average predict index $= 1.855$	

Table 1 and 2 are test results of filtering and smoothing using threshold of 5; Table 3 and 4 are test results of filtering and smoothing using threshold of 10. (name$_i$ is artist name in Japanese/Chinese) When $h = 5$, both filtering and smoothing have similar mean, but smoothing has higher median. When $h = 10$, there is not too much difference, but smoothing still performs better. Looking at the distribution plots and these statistics, we can guess that smoothing fits model in such ways that overall performance is much higher, but it sacrifices prediction accuracy for a small number of users. On the other hand, filtering fits model that can make at least some predictions for all users, but performs slightly worse than the smoothing case overall.

Table 3. Test results of filtering method with $h = 10$

Mean accuracy $= 0.731$	
Median accuracy $= 0.788$	
g_l with highest hit frequency	rock, metal, indie, pop, christmas, synthpop, polish, jazz, electronic, hip
g_l with highest miss frequency	reggae, rock, hip, metal, pop, jazz, japanese, anime, electronic, house
a_m with highest hit frequency	The Knife, Nine Inch Nails, The 69 Eyes, The Beatles, Radiohead, Metallica, Pink Floyd, Evanescence, Coldplay, Placebo
a_m with highest miss frequency	name$_1$, name$_2$, M-Flo, Soda Stereo, Bonnie Pink, Bob Marley, L'Arc En Ciel, Los Fabulosos Cadillacs, name$_3$, Los Piojos
Average predict index $= 2.364$	

Table 4. Test results of smoothing method with $h = 10$

Mean accuracy $= 0.739$	
Median accuracy $= 0.874$	
g_l with highest hit frequency	rock, metal, indie, pop, christmas, synthpop, polish, wave, electronic, industrial
g_l with highest miss frequency	rock, reggae, metal, pop, hip, jazz, indie, classical, japanese, house
a_m with highest hit frequency	The Knife, Nine Inch Nails, The 69 Eyes, The Beatles, Metallica, Radiohead, Pink Floyd, Evanescence, Coldplay, Pearl Jam
a_m with highest miss frequency	name$_1$, name$_2$, M-Flo, Frederic Chopin, Soda Stereo, Bonnie Pink, L'Arc En Ciel, Bob Marley, Tenhi, Los Fabulosos Cadillacs
Average predict index $= 2.344$	

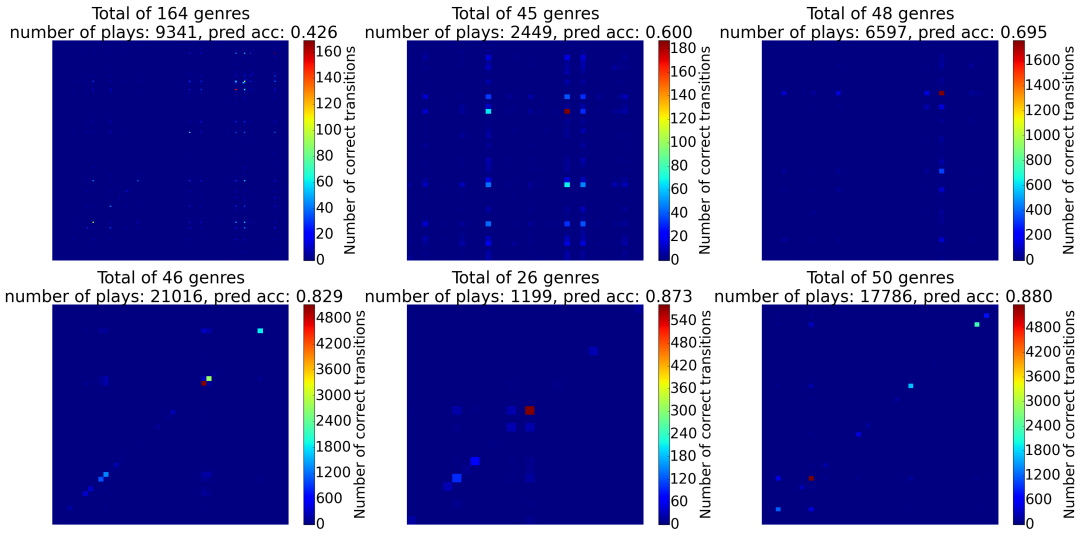

Figure 9. Heatmap of frequencies of genre transition that resulted in correct prediction for 6 users in filtering inference using $h = 10$. Redder the color is, higher the number of correct transitions is.

First row of genres in the tables is a list of genre predictions made by HMM that had the highest number of matches; second row is a list of genre predictions that had the highest number of mismatches. We see that popular genres, such as rock and metal, are in many of these lists. This is not surprising as it is the hidden state that occurs the most; these states show up frequently, and thus have many possible likely movements onto different states. Assuming that our model performs correctly, we can also safely make assumptions that the genres with highest hit frequency are genres that are liked by users who fit our model well, and the genres with highest miss frequency are genres that are liked by users of unique genre preference trend that is hard to capture with HMM.

We have similar sets for artists. We see some popular artists in group of artists with highest hit frequency, such as Nine Inch Nails and The Beatles. One interesting thing to note is that there are some artists that are popular, such as Chopin and Bob Marley, in the group of artists with highest miss frequency. The reasons for these artists having high miss frequency seem to be similar reasons that popular genres had high miss frequency: many people with many different genre trends listen to the artist, and divert to many different genres afterwards, which can be hard to capture.

The average predict index is the average of all hit prediction index $I_{u,t}$, which is the index of predicted values that match the true genre for each correct prediction, for user u at time t. Therefore, lower average predict index indicates a better model, for a fixed h. It is interesting to see that average index of filtering is smaller than that of smoothing when $h = 5$, even though smoothing performed better. This is because smoothed marginals are skewed and contain many zeros.

The Tables above show differences in two approaches. Filtering performs less accurately, but its distribution of prediction accuracy values is more spread out and has values significantly greater than zero for all users. On the other hand, smoothing performs better overall; however, it fails almost completely for a small group of users. This indicates that smoothing process could overfit the model on training dataset.

Figure 9 shows heatmaps indicating number of transitions from a genre to another that led to a correct match, based on filtering inference using $h = 10$. Figure 10 shows the same heatmaps for smoothing, using $h = 10$. If a cell (i, j) has color closer to red, many transitions from g_i to g_l has made correct prediction, compared to other transitions. We see that users with high prediction accuracy have simple taste in music, and like to transit from a genre to the same genre, as many of the diagonal cells have high number of correct transitions, which indicate times in which the user did not change the genre of the music.

We can view each distinctive ”rows” and ”columns” of brighter colored cells as major genres that the user listens to. (that was predicted correctly) Excluding cases in which prediction accuracy is poor, (the first plots of Figure 9 and 10) we see that users with more complex taste who like to jump from a genre to many genres are harder to predict.

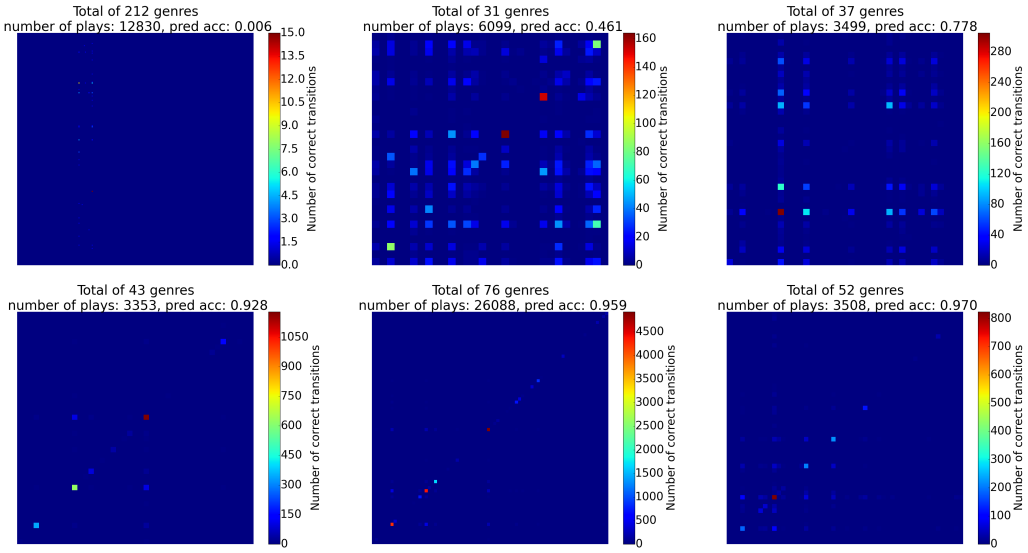

Figure 10. Heatmap of frequencies of genre transition that resulted in correct prediction for 6 users in smoothing inference using $h = 10$. Redder the color is, higher the number of correct transitions is.

In other words, as the number of "major genres" increases, prediction accuracy tends to decrease. It is also noteworthy that filtering managed to catch a small pattern of genre transition, as seen in the first plot of Figure 9, smoothing failed almost completely for a small number of dataset as seen in the first plot of Figure 10.

This may indicate that this model, especially the smoothing method, is somewhat overfitted on the users who have simple taste, who make minimal amount of transitions from a genre to another. Therefore, our model performs not so greatly with users who make many genre transitions. This leads to a potential improving point of this model; we can train different HMM based on the distribution of genres listened to. If a user's genres played are focused on a few points or small in numbers, we train/test the user data using HMM trained for users with simple genre taste; if a user's genres played are more dispersed or large in numbers, we train/test the user data using HMM trained for users with more exploratory attitude towards genres.

4.5.5. Problems

There is one problem that must be fixed in order to maximize the predictability of the model. As mentioned in section 4.3, due to the immense number of artists, there are often data points in test set that contain artist that has not been seen during the training phase. Then our class-conditional distribution $\boldsymbol{\psi}_t$ is all zeros and this causes the smoothed/filtered marginals to convert to zero, returning same prediction value after it converges to zeros.

There are two ways in which we tried to fix this. First, we considered skipping the play record at time t, if x_t was not observed in training set. However, this could pose some problems as this would interfere with the Markovian properties of the model. If we skip time t because x_t was not observed in training phase, the Markov transition probability from state z_t to z_{t+1} is also skipped, disrupting the Markovian chain of the hidden states.

The other method that was considered, which was implemented to get the results above, is to replace $\boldsymbol{\psi}_t$ with vector of ones. This does not seem to heavily affect the result of our model, if at all. However, mathematical rigorousness of this step needs to be proved so that we can use it safely.

5. Conclusion

5.1. Summary

We investigated ways in which we can fit a Hidden Markov Model onto a set of artist-future genre pairings extracted from last.fm 1K dataset. Different ways in which data was preprocessed, and shortcomings of this dataset have been discussed. Using a training set of 400 users, we fit parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\Psi}, \{\boldsymbol{\psi}_t\}_{1:T})$ using MLE methods. After parameters have been fit, we can use inference methods such as filtering and smoothing methods; Viterbi algorithm also should to be debugged and compared to other methods. We obtain better results with smoothing method, as

it takes future information into account when calculating the marginals. Heatmap analysis shows patterns in varying degrees of preferred genres of the users.

5.2. Further Work

5.2.1. Dataset Issues

It is not clear how much last.fm's recommendation system affects these results. In the time when this dataset was taken, recommendation systems would likely suggest songs by the same artist or from the same album, which is what preliminary analysis uncovered.

The problem of selecting the best genre tagging method, as discussed in section 4, is a big factor that affects our model. Using genre similarity scores returned by EchoNest, we hopefully can classify or cluster genres into a smaller number of macro-level genre groups. Last.fm's API also has such method, but the method does not work, and it seems weird as it is the only method that returns nothing.

Other major problem is the method of labeling songs that do not have any genre data. One way of addressing this problem is to estimate the labels as best as we can using methods such as clustering, binning, brute-force, etc. Another way is to simply use EM algorithm to fit an unsupervised HMM, then use the top genre labels to estimate a good starting point for Baum-Welch algorithm.

5.2.2. Models and Algorithms

Viterbi algorithm, once completely debugged, could return an interesting result that differs from filtering or smoothing inference results. Unlike filtering and smoothing methods that use MLE to estimate the parameters, Viterbi algorithm does it by finding the most likely sequence of hidden states.

Implementing a Baum-Welch algorithm to fit the optimal number of hidden states, rather than using the large set of genre labels, could return a model that fits the data better. Many of our current genre tags have varying degrees of similarities with other genre tags, and estimating the best hidden states would reduce computation time and lower the dimensions of parameters to be fitted.

References

Bertin-Mahieux, Thierry, Eck, Douglas, Maillet, Francois, and Lamere, Paul. Autotagger: A model for predicting social tags from acoustic features on large music databases, 2010.

Celma, O. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

Eck, Douglas, Bertin-Mahieux, Thierry, and Green, Stephen. Automatic generation of social tags for music recommendation. *NIPS*, 2008.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

Inoue, Kohei and Urahama, Kiichi. *Fuzzy Clustering Based on Cooccurrence Matrix and Its Application to Data Retrieval*, volume J83-D-II. Denshi Joho Tsushin Gakkai Ronbunshi, 2001.

Murphy, Kevin. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Piantadosi, Steven T. Zipfs word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, 2014.

Rabiner, Lawrence R. A tutorial on hidden markov models and selected applications in speech recognition. IEEE.

Xi Shao, Changsheng Xu and Kankanhalli, Mohan S (eds.). *Unsupervised Classification of Music Genre Using Hidden Markov Model*, ICME, 2004. IEEE.