The Next Best Songs
Spotify Song Recommendation

Alexander Bricken
CS146 & CS156
Minerva Schools at KGI

Table of Contents

Abstract

Music is an important component of our daily lives. We dance, sing, enjoy, and cry simply because of music. But what music is right for any given moment? Spotify tries to use song and playlist data to predict this. In this paper, I provide a preliminary model for predicting the next best song for a music playlist on Spotify. Specifically, I apply a blend of supervised and unsupervised machine learning methods (including K-nearest-neighbours, neural networks, multi-logit regression, and gaussian mixture models) to find songs that are most similar to a playlist's features. The advantages and disadvantages of my approach are discussed, and my final results are displayed. In conclusion, I realise that there are a few fundamental flaws with my approach, however, the results are somewhat promising and provide unique insight into a variety of machine learning and Bayesian methods for predictive inference.

Introduction & Problem Definition

A playlist in Spotify is a compilation of the songs a user loves, generally created to reflect a certain genre or mood of music. Just as a DJ tunes music to appeal to the audience, Spotify uses vast amounts of data and machine learning algorithms to seamlessly play the "next best" song once a user finishes the playlist they are listening to. Typically, this song is predicted using features such as the name of the playlist, song traits, and the similarity of preferences across users with corresponding taste (via methods like collaborative filtering).

In this research, I build a preliminary model for understanding and challenging the Spotify music recommendation system. Using the Spotify Million Playlist dataset (Spotify, Spotify Million Playlist Dataset Challenge, 2020) and Spotify API (Spotify, Explore, 2020), I query the features of songs contained in a variety of playlists. From there, I generate my own list of songs from a variety of genres, and finally predict which of these songs has the highest probability of a user liking it once their respective playlist has been played. This task can be summarised as: "what are the next best songs given a playlist?"

My hypothesis is: "Can a computational model predict, to a good quality, the next best songs for a playlist by using the mean of its songs' features and finding the similarity of such features to songs in a high dimensional space?" The computational approach is thus two-fold: I use three different classification methods (KNN, neural networks, and Bayesian logistic regression) to perform a supervised task of using the labelled song genres to predict the genres of my playlists. From there, I use a Gaussian Mixture Model clustering method to represent the clusters of potential songs within a genre that a playlist (already classified to that genre) would be most similar to. By finding the most similar songs in a high dimensional space represented by the features queried through the Spotify API, we can successfully predict the "next best" song for a playlist.

I am motivated to pursue this computational technique over others (such as collaborative filtering and more complex machine learning models) because of the novelty of its predictive method (we use a blend of supervised and unsupervised methods to build, in essence, a recommendation system) and the ease of comprehension. While we cannot visualise a high dimensional space to represent all the features of a song at once, we can still think about the "similarity" of a playlist and song as just the Euclidean distance between the two in this space.

The remainder of this report is organized in the following way: (i) Method & Solution Specification, where my data pre-processing and general approach is discussed, and where I distinguish my machine learning (CS156) and Bayesian (CS146) approaches; (ii) Results,

where I show the output of my methods; (iii) Testing & Analysis, where I discuss my results and the advantages and disadvantages of my approach; (iv) Conclusion, where I summarise my findings; (v) References; and (vi) Appendices, where I link to my code and additional figures.

Method & Solution Specification

Data Uploading & Pre-processing

Initialisation. This is where the libraries necessary for my initial data exploration and pre- processing are imported. As well as this, I define the necessary keys needed and initialise my use of the Spotify API.

Data Upload. Using the Million Playlist dataset downloaded 1000 example playlists to start with. From there, I unpacked the JSON files into lists containing the respective song URIS for each playlist. I made sure to define functions here so I could vary the size of unpacking without re-running all cells.

Variable Selection. The Spotify API provides a lot of information on a given playlist, such as its name, the number of followers it has, how many tracks it contains, whether or not it is collaborative, and the tracks within it all have a variety of features. When querying the API, I made sure to only query the necessary features for song prediction. In general, this means a variable that is measurable, independent, and is a float, integer, or binary value. Features like "liveness" which was a float that denotes the "background audience" of a song, telling us whether or not the track is performed live or not, I deemed not important for song prediction and hence did not store them. However, aspects of songs like "danceability", "speechiness", and "valence", were all extremely important. Overall, I used 9 features as variables for representing my playlists and songs.[1]

Generating DataFrames. My playlist dataframe I generated by taking the mean of all features of songs in a playlist, where each row represents a playlist:

	pid	key	acousticness	danceability	energy	instrumentalness	loudness	speechiness	valence	tempo
0	834000	0.645308	0.173219	0.857485	0.608134	0.015506	0.806802	0.209173	0.326380	0.395954
1	834001	0.757040	0.451602	0.480572	0.532890	0.389105	0.654947	0.010219	0.499174	0.353239
2	834002	0.679062	0.098866	0.760773	0.782856	0.010693	0.903573	0.048531	0.634635	0.427712
3	834003	0.567796	0.010496	0.433334	0.965381	0.000039	0.960515	0.059296	0.593296	0.697373
4	834004	0.631160	0.413368	0.551870	0.498053	0.027596	0.710757	0.009369	0.387782	0.449970
...	...	...	...	...	...	...	...	...	...	...
495	834495	0.652280	0.291041	0.528743	0.521976	0.043901	0.686463	0.005384	0.412935	0.413829
496	834496	0.390438	0.380231	0.456496	0.652084	0.000217	0.818805	0.055452	0.462519	0.704571
497	834497	0.482072	0.375387	0.580396	0.662425	0.134960	0.782003	0.041677	0.553171	0.431537
498	834498	0.668866	0.220879	0.600123	0.736781	0.029747	0.852047	0.077878	0.539488	0.491324
499	834499	0.506490	0.037309	0.424949	0.906229	0.026403	0.903016	0.036353	0.578541	0.621498

[1] **#variables:** only by identifying the relevant features for evaluating my hypothesis could I successfully code and trust my models. I made sure to consider the values of my features, that were primarily floats, and their meanings. As well as this, I cut out unnecessary features such as liveness. Finally, I considered the confounders of my dataset, such as there being more than 10 genres on Spotify.

Because I was not using the playlist dataframe for my supervised learning task of training models, I did not need a large amount. Hence, I imported 500 playlists and calculated their mean features.

For generating my song dataframe, I used the Spotify API to query the top songs from 10 different genres. I selected a variety of the most popular genres to get a diverse range of features and to increase the pool of songs I could sample from. The genres selected are: 'pop', 'hip-hop', 'edm', 'latin', 'rock', 'r-n-b', 'country', 'jazz', 'classical', and 'alternative'. I label encoded these genres for ease of use in my models. Also, to overcome a constraint on how many times I could query the Spotify API, I built an algorithmic solution that would allow me to query it in multiple batches. The problem with this however, is that I had the potential to query the same song twice. Hence, using the set() function I filtered out duplicates. In the current code, I made 20,000 song requests, and only 5,201 of those were valid due to duplicates

	uri	genre	key	acousticness	danceability	energy	instrumentalness	loudness	speechiness	valence	tempo
0	spotify:track:1snNAXmmPXCn0dkF9DaPWw	pop	0.909091	0.383532	0.586949	0.393220	0.000115	0.857130	0.005014	0.234933	0.166156
1	spotify:track:7i2DJ88J7jQ8K7zqFX2fW8	pop	0.545455	0.107426	0.671537	0.587109	0.000001	0.903802	0.021227	0.439992	0.376020
2	spotify:track:6n4U3TIzUGhdSFbUUhTvLP	pop	0.181818	0.087948	0.931891	0.635329	0.000003	0.897210	0.269597	0.385864	0.591400
3	spotify:track:6xbraxG0bSBOuAr5lUXmtM	pop	0.000000	0.385540	0.729759	0.656426	0.000000	0.858336	0.460137	0.677319	0.615619
4	spotify:track:74vs7PD7iG2GnWUGnb8Yuq	pop	1.000000	0.003711	0.581457	0.751863	0.000000	0.916288	0.059001	0.387946	0.496441
...	...	...	...	...	...	...	...	...	...	...	...
5196	spotify:track:2jZgzSxNSg1hTCq0ewWHGJ	alternative	0.181818	0.021281	0.599033	0.897531	0.000000	0.900576	0.023901	0.912564	0.586016
5197	spotify:track:5u4q8PgOpFOa5tG4yrFAzw	alternative	0.181818	0.012646	0.739646	0.618251	0.283838	0.857385	0.015377	0.422296	0.435969
5198	spotify:track:6EvgWSbWmQelxrQBmhEdXh	alternative	0.090909	0.077607	0.473800	0.780997	0.008707	0.854902	0.053652	0.264078	0.314157
5199	spotify:track:5Z7cI9glyUTDTRtWSs8K9I	alternative	0.454545	0.042968	0.622103	0.625283	0.058182	0.758425	0.071536	0.178724	0.579858
5200	spotify:track:0MTHQlWgo6iCNjXgM5Z9fr	alternative	0.181818	0.469877	0.351862	0.296779	0.001677	0.716093	0.021728	0.032685	0.410515

Finally, when generating these dataframes, I decided to normalise the variables as well, so that the machine learning models could best interpret them with equal weighting and appropriate distribution. This was done using the MinMaxScaler function from sci-kit learn.

Machine Learning Approach. As mentioned in the introduction, my approach for finding the next best songs was two-fold: using supervised methods, train models (KNN, neural network, and Bayesian logistic regression) to predict the genre label of a given playlist. From there, use unsupervised clustering (Gaussian Mixture Model) within that genre to find the songs with the closest Euclidean distance in a high dimensional space to that song. These songs are hypothesized to be the likely “next best”.

CS156 – Machine Learning

K-Nearest Neighbours

Now that the data was in a usable format, I built machine learning models to train on the genre labels of our song dataframe and predict the genre labels of our playlist dataframe. I could measure the accuracy of my models by using a train-test-split approach.

I started by implementing a K-nearest neighbours (KNN) algorithm. This is easy to implement and provides a strong baseline for our future models. The assumption of KNN is that similar data points exist in close proximity in a space. This falls perfectly in-line with my hypothesis. The value of K is an indicator for a specific number of samples that the algorithm should classify as groups. The most frequent label (genre) within these groups will be the label for all of them. Hence, to predict using KNN I simply put a playlist in that space with classified groups, and whichever group it falls into is the label it receives. I had to be careful because KNN can suffer from the curse of dimensionality, which occurs when considering too many features. However, in this case I only have 9, which means the model likely doesn't suffer from the curse of dimensionality.

For finding the best value of K, I built an algorithm that iterates through values of K ranging from 1 to 50, and uses 10-Fold cross-validation to find the average accuracy for each K. By doing so, I can find the best value of K and validate that value using our cross-validation, such that we are not overfitting to a single training dataset. The following is a plot of my results:

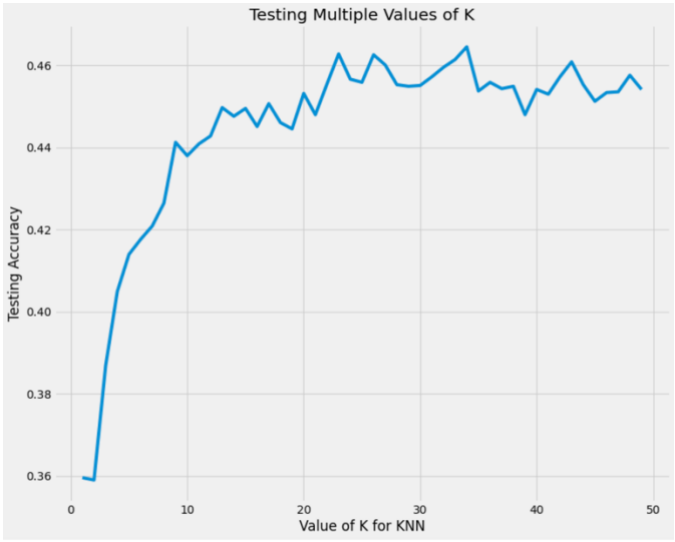


Fig 1. The average accuracy of our model across multiple values of K.

To visualise this process, I used Principal Component Analysis to reduce the dimensions of my data to 2 components. While the explained variance is low (62.9%) and therefore we are not accurately representing all of our features with just 2 components, it is a good way to visualise our KNN algorithm (despite not using it for my finalised KNN classification):

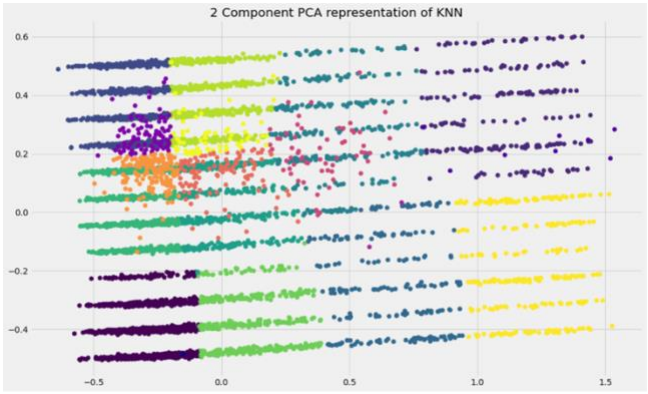


Fig 2. A 2-component representation of our KNN predictions.

Neural Network

I built an 8-layer neural network with maximum width of 96 neurons. My goal for implementing this neural network was to achieve a higher classification accuracy than my KNN model.

Using Keras features, I implemented some unique tuning to optimize the classification accuracy of my neural network. Firstly, I used 'Dropout' layers (Keras, 2020) to prevent overfitting. This applies to the training process of my neural network and randomly sets input units to 0 with a frequency of 0.1 at each step during training time. This acts as a form of regularization to temporarily remove neurons from the forward pass and not update weights on the back propagation, making the model less sensitive to specific neuron weights and more generalizable (Brownlee, Dropout Regularization in Deep Learning Models With Keras, 2016).

My ReLU (Rectified Linear Unit) activation layers act as the default neurons in my neural network. The function "is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero" (Brownlee, A Gentle Introduction to the Rectified Linear Unit (ReLU), 2019).

For concision purposes, I will not go into the specific details of my model further than briefly mentioning the other features I implemented. A learning rate for the Adam optimizer allowed me to control how quickly the model is adapted to the problem. Label smoothing allowed me to make my model less overconfident in its predictions. This regularization method allowed the model to not "overclassify" a playlist – but rather restrains the largest

logit from becoming much bigger than the rest. This allows the model to think about different genres and a combination of genres, rather than being overconfident towards one. Finally, an early stopping method allowed me to prevent the model overfitting by running too many epochs.[2]

I measured the accuracy and loss of my model using the following graphs and they helped me consider the overfitting/underfitting of my model to balance the bias-variance trade-off by tuning my hyperparameters (train model loss should never go to 0, otherwise we are overfitting).

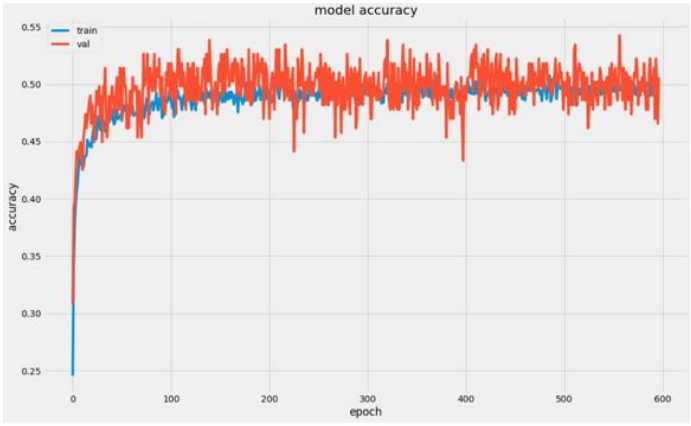


Fig 3 & 4. The model accuracy and model loss plots used to evaluate my model's performance.

CS146 – Bayesian Inference/Computational Statistics

Bayesian multivariate logistic (multi-logit) regression is another method that would allow me to classify my playlists into genres. In my approach, I calculate a posterior distribution over all genres for each playlist. I therefore know the probability of a given playlist being classified as any given genre. In simple Bayesian terms, we can write this as:

$$P(\text{genre} \mid \text{playlist}) = \frac{P(\text{playlist} \mid \text{genre}) \times P(\text{genre})}{P(\text{playlist})}$$

[2] **#algorithms:** I analysed my algorithmic strategy throughout this assignment and did my best to explain my code and understand the special features I implemented on my neural network. As well as this, I wrote thorough comments in my code and abstracted a lot of the computation through use of functions.

In general, as in univariate regression, we can write $\pi(x)$ to represent the probability of an event that depends on k covariates. We know this can be written, using the logit function, as:

$$\pi(x) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p}}$$

But here we have multiple covariates instead of one.

As we have not gone into the details of Bayesian multivariate logistic regression in class, much of my interpretation comes (Sean M. O'Brien, n.d.) and the Stan documentation (Stan, 2020). The model can be written as the following:

$$y_{ik} = (y_{ik}, \ldots, \mathrm{y_{nk}})' \sim \text{ Multivariate Logistic } (\mathbf{X}_i\beta, \mathbf{R}_i)$$

where the song predictions are stored in a vector or matrix y_{ij} and β is our logistic regression coefficient matrix. To compute the posterior in this way, I use Hamiltonian Monte Carlo importance sampling implemented using the PyStan library in Python. This allows ne to calculate expected parameters over a population of the form:

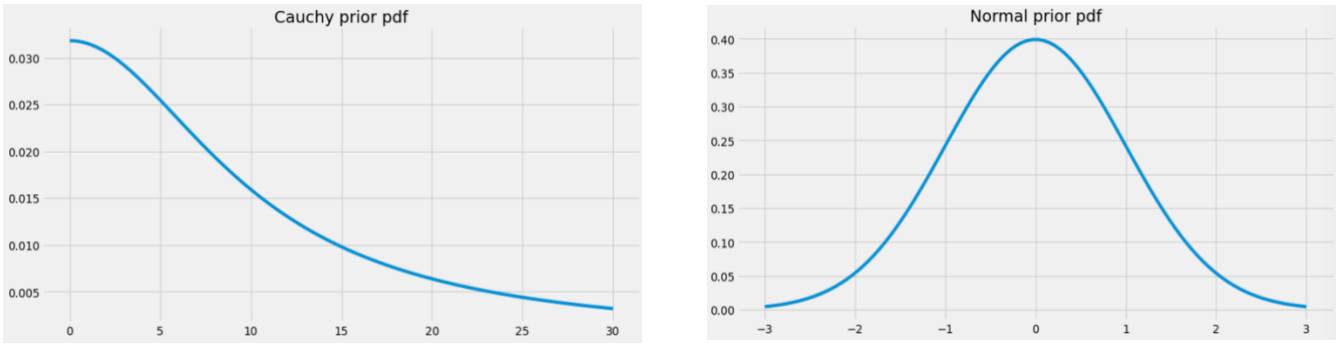
$$E = \int_{-\infty}^{\infty} g(x)\pi(x)dx$$

In my Stan model, I calibrate the K possible genres for each output variable y_n. I also calibrate an N, D-dimensional matrix that contains the features of my song dataframe for y_n. I generate a prior for Beta, my regression coefficient matrix with a Gaussian distribution: $Normal(0,1)$. I use this because a Gaussian is moderately tailed, a center of 0 is right because the coefficients can be negative, and my standard deviation is 1 because all of my values are normalised between 0 and 1. I also model a $Cauchy(0,10)$ prior for the intercept of my multivariate logistic regression which is used to predict y_{pred}. I do this because it is a "weaker prior" (heavier tails), and flat-tailed distributions generally allow for robust inference and easy and stable computation in logistic regression by placing iteratively weighted least squares within an approximate EM algorithm (see Figs. 5 & 6) (Gelman, 2008).

The categorical-logit distribution applies a softmax function, which is a generalization of the logistic function to multiple dimensions, internally to calculate y_n which is a simplex containing the trained probabilities for the model matrix x, taking into account the coefficient matrix. To predict the genres of the playlist dataframe, I use the typical logistic regression formula but querying each row of the x matrix and each column of coefficients from the β

coefficient matrix, instead of individual values, and taking the dot product. I also use my alpha parameter as the intercept modelled on the Cauchy prior as discussed before:

$$y_{ik} = \alpha + \sum_{i=1}^{D} \beta_{ik} * x_i$$



Figs 5 & 6. Plots of the priors used, as discussed above.[3]

To represent the Bayesian multivariate logistic regression as discussed above, a factor graph is drawn. The graphs of the model and the generated quantities (predictions) are separated.

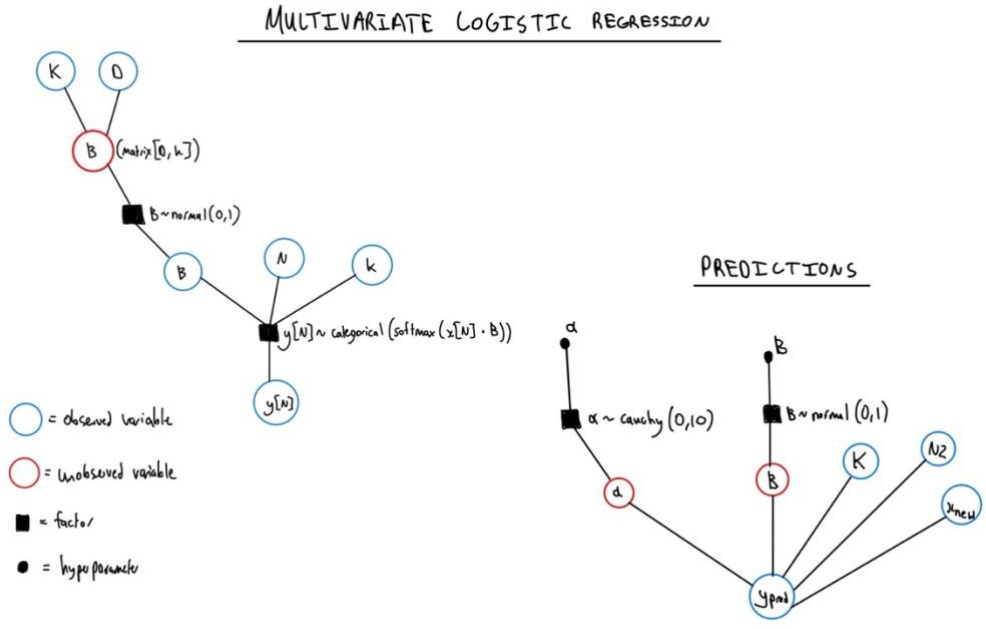


Fig 7. Two factor graphs showing the PyStan Multi-Logit classification model.

[3] **#distributions:** I selected and justified my use of the prior distributions to an appropriate level of detail. I made sure to cite the research I had looked out to guide these decisions as well. I also effectively sampled from these distributions in my PyStan model.

CS156 & CS146 – Gaussian Mixture Model

The unsupervised portion of my approach is the Gaussian Mixture Model (GMM) which is only applied after we have classified playlists into genre. I use a GMM because it can successfully find a probabilistic representation of a playlist in a range of clusters, each, in theory, containing similar songs from the specific genre the playlist has been classified to. The advantage of using a GMM instead of K-means clustering, both of which are generally easy-to-apply unsupervised models, is that our GMM can handle non-circular clusters of data, as I have specified using the "full" covariance type. The second advantage is that a GMM performs soft-clustering, telling us the probabilities that a given playlist belongs to each of the possible clusters. This is useful for finding songs that are similar but outside of the cluster assigned (which may be necessary if the songs in the cluster a playlist is defined to run out).

Mathematically, we can write the likelihood that any given sample came from a Gaussian k in our GMM as

$$p(z_i = k \mid \theta)$$

where θ represents the parameters of our Gaussian (mean, covariance, weight).[4] Similarly, we can write the likelihood of observing a data point given that it came from our Gaussian k as

$$N(xi \mid \mu_K, \epsilon_k)$$

To take into account all possible distributions, we can simply use the sum rule, and marginalise over all other samples under the assumption that they are independent of one another (Maklin, 2019). We use the log likelihood here because the logarithm of a product is the sum of the logarithms:

$$\log\left(p(x \mid \theta)\right) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{D} \left(N(x_i|\mu_k, \epsilon_k)\pi k\right)\right)$$

In order to calculate the parameters of our Gaussians, we use the Expectation Maximisation algorithm, which helps us find the local maximum likelihood estimates of our parameters. To summarise this process, iteratively the EM algorithm performs an expectation (E) step, "which creates a function for the expectation of the log-likelihood evaluated using

[4] **#probability:** thinking and writing about Bayesian probabilities, likelihoods, and the expectation maximisation algorithm allowed me to apply this HC. I tried to explain with as much detail as necessary without getting caught in the details of the maths.

the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step" (Wikipedia, 2020).[5]

The results of my GMM implementation can be seen in the next section.

[5] **#modeling:** Stan implements a Hamiltonian Markov Chain Monte Carlo method. I used this model to output predictions and tried to explain with parsimony how this works. Also, I evaluated the effectiveness of all of my models in the Analysis section and provided effective solutions to some of the disadvantages of my approach.

Results

CS156

The KNN algorithm has an accuracy score of 43.2%. This is just over 4 times more accurate than if were were to choose genre at random.

To assess the performance of the model, a confusion matrix was used:

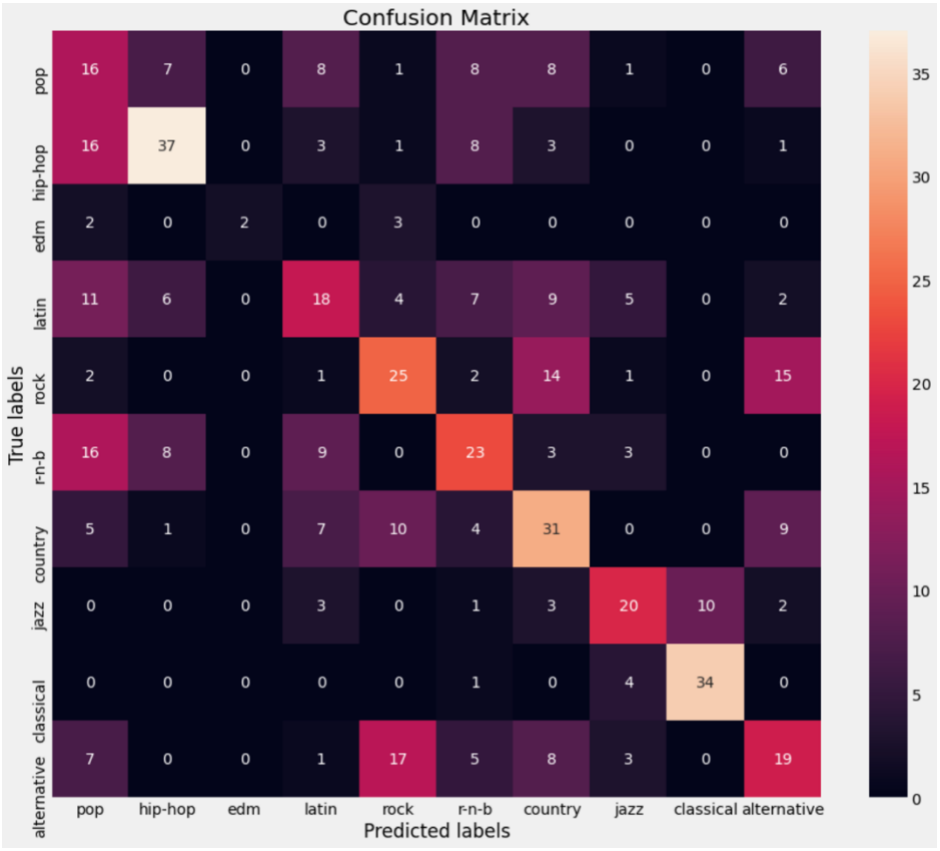


Fig 8. A confusion matrix to assess the accuracy of the KNN model. I can see that a few genres like classical and hip-hop are easier to classify than others, like alternative and latin.

Using this KNN model, I could predict the genres of playlists and output an array of 500 labels, corresponding to the 500 playlists from the dataset.

After implementing the Gaussian Mixture Model, I use the KNN model predictions for the genres of playlists and then find the corresponding songs within that genre that are

best suited to the playlist from the GMM cluser. For example, taking the first playlist in the dataset called “pump”, this is the output (see Appendix for more predictions):

```
Name of playlist: pump
The playlist is genre: pop


The recommended songs, in no particular order, are:
3005, by Childish Gambino
Don't Let Me Down (feat. Daya), by The Chainsmokers
Send My Love (To Your New Lover), by Adele
Heart Attack, by Demi Lovato
Wake Up, by Fetty Wap
We Are Young (feat. Janelle Monáe), by fun.
Stressed Out, by Twenty One Pilots
Someone New, by Hozier
Sucker for Pain (with Wiz Khalifa, Imagine Dragons, Logic & Ty Dolla $ign feat. X Ambassadors), by Lil Wayne
You Da Baddest (feat. Nicki Minaj), by Future
Famous, by Kanye West
Don't Mind, by Kent Jones
Power (feat. Stormzy), by Little Mix
To the Max (feat. Drake), by DJ Khaled
Wet Dreamz, by J. Cole
```

For the neural network, I also used a confusion matrix to examine the results of my song genre predictions.

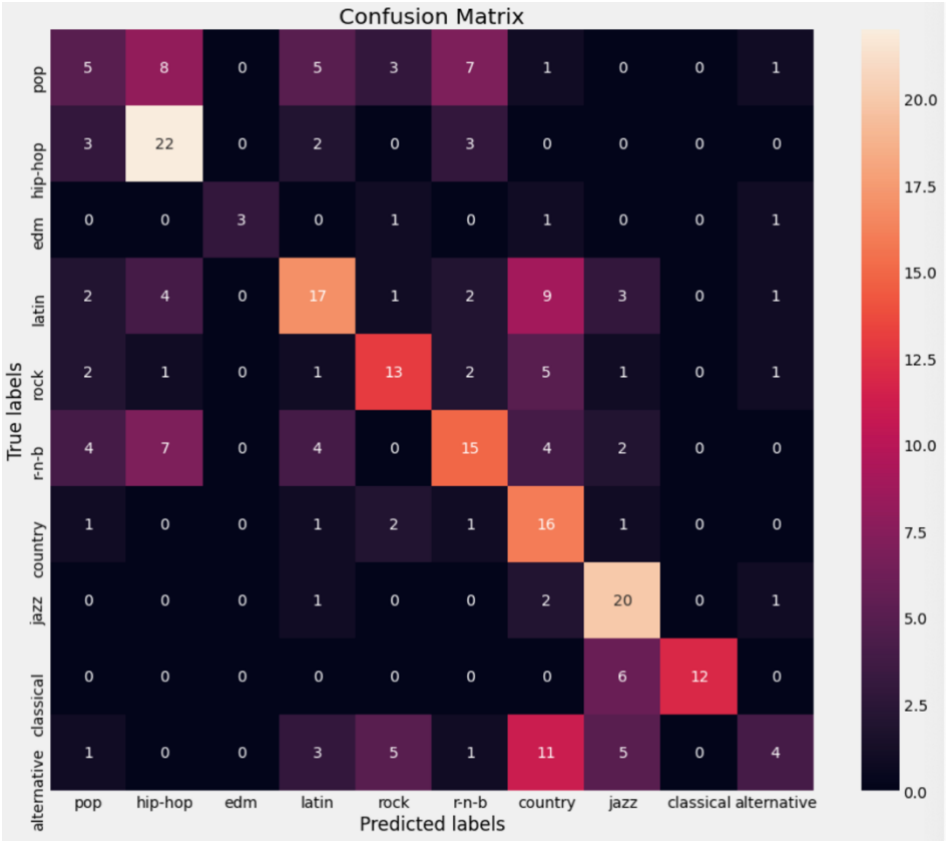


Fig 9. A confusion matrix to assess the accuracy of the neural network. Interestingly, more songs are being classified as country as we would expect.

The accuracy of the neural network was 48.6% when testing on our test dataset after train-test-splitting. Like the KNN algorithm, the neural network can make predictions on the playlist dataset.

As mentioned above, once we implement the GMM model on our genre predicted playlists, we can output the recommended songs. We use the “pump” playlist again here for comparison:

```
Name of playlist: pump
The playlist is genre: hip-hop


The recommended songs, in no particular order, are:
Glow Like Dat, by Rich Brian
REEL IT IN, by Aminé
Sippin On Some Syrup (feat. UGK (Underground Kingz) & Project Pat), by Three 6 Mafia
U Can't Touch This, by MC Hammer
The Breaks, by Kurtis Blow
Swang, by Rae Sremmurd
I'm N Luv (Wit a Stripper) (feat. Mike Jones), by T-Pain
Collard Greens, by ScHoolboy Q
Ladders, by Mac Miller
D.R.U.G.S., by Ab-Soul
No Rest, by Lil Skies
Day 'N' Nite (nightmare), by Kid Cudi
```

Interestingly, the genre is classified differently. The reason for this is explored in the Analysis section.

CS146

Like our machine learning models, we can evaluate the performance of the classification of our Bayesian Multivariate Logistic Regression by looking at our results. The n_eff (number of effective samples) for our model across all predictions is significantly high (over a couple thousand). Hence, we can assume our samples are independent from Stan’s Hamiltonian Monte Carlo method. Likewise, all of our Rhat values are equal to 1. This suggests our algorithm has converged and our Markov chains are mixing well.

We can display a sample of our y_pred values in a scatter plot like the following:

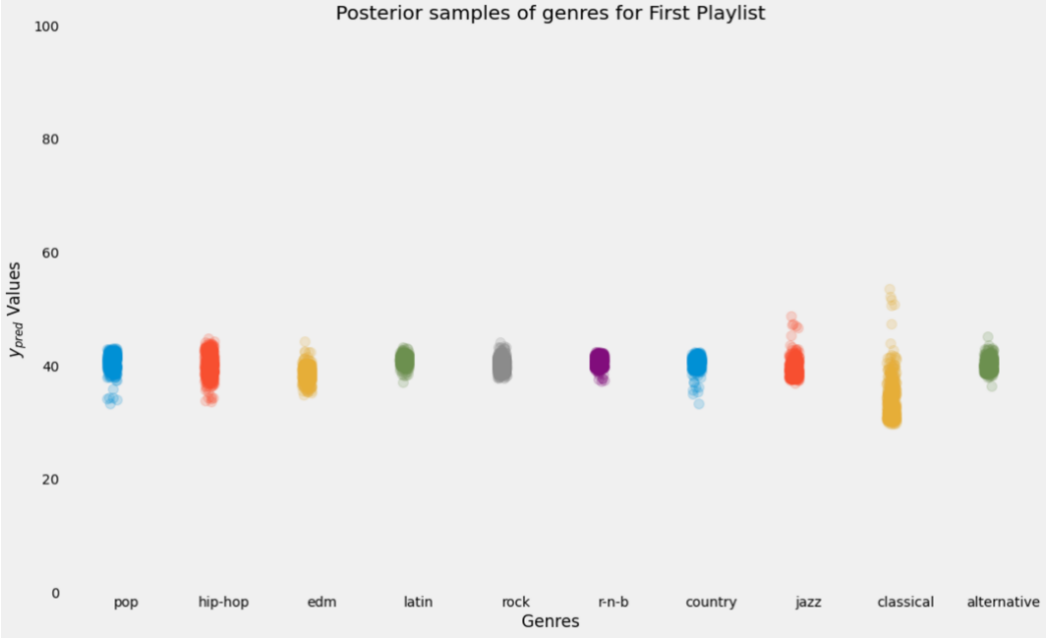


Fig 10. Scatter plot to show the posterior samples for the first playlist. The y-axis is not easily interpretable because we have not converted the logit to a probability. Still, by eye-balling it we can see that hip-hop is the likely genre for this playlist.

Like our machine learning models, we can apply our list of predicted genres for our playlist by using our Gaussian Mixture Model. The result on our first playlist are as follows:

```
Name of playlist: pump
The playlist is genre: hip-hop


The recommended songs, in no particular order, are:
Glow Like Dat, by Rich Brian
REEL IT IN, by Aminé
Sippin On Some Syrup (feat. UGK (Underground Kingz) & Project Pat), by Three 6 Mafia
U Can't Touch This, by MC Hammer
The Breaks, by Kurtis Blow
Swang, by Rae Sremmurd
I'm N Luv (Wit a Stripper) (feat. Mike Jones), by T-Pain
Collard Greens, by ScHoolboy Q
Ladders, by Mac Miller
D.R.U.G.S., by Ab-Soul
No Rest, by Lil Skies
Day 'N' Nite (nightmare), by Kid Cudi
```

and for a randomly selected playlist,

```
Name of playlist: Together
The playlist is genre: pop


The recommended songs, in no particular order, are:
You & Me - Flume Remix, by Disclosure
Let Her Go, by Passenger
Hear Me Now, by Alok
Crush, by Yuna
Controlla, by Drake
7 Years, by Lukas Graham
See You Again (feat. Charlie Puth), by Wiz Khalifa
Come and See Me (feat. Drake), by PARTYNEXTDOOR
Psycho (feat. Ty Dolla $ign), by Post Malone
Say It, by Tory Lanez
Don't, by Bryson Tiller
Plug Walk, by Rich The Kid
Cool Girl, by Tove Lo
Lucid Dreams, by Juice WRLD
Weekend (feat. Miguel), by Mac Miller
Homemade Dynamite, by Lorde
Don't Matter To Me (with Michael Jackson), by Drake
```

Analysis & Conclusion

The reason I took the two-fold supervised to unsupervised approach is to narrow the unsupervised scope of the problem down. To predict the next best songs for a given playlist without even knowing the genre of that playlist would be too difficult with my approach of using a high dimensional space and the Euclidean distance between points calculated via the Gaussian Mixture Model.

To compare the output of my models, I defined functions to calculate the similarity in predictions. 40.4% of all three of the models have shared labels. When compared as couples, the KNN and neural network predictions are 50.4% the same, the KNN and Bayesian multi-logit predictions are 54.2% the same, and the neural network and Bayesian multi-logit predictions are the most similar at 58.6% shared labels. This variation in model performance suggests no two models are extremely similar. This is likely because of the different constraints of models. Our best performing model is the neural network, which was the most difficult to implement and is more at the cutting edge of machine learning right now. On the other hand, the KNN performed relatively well, and it is hard to evaluate the performance of the logistic regression as a Bayesian model. However, the KNN is constrained by being sensitive to noisy data as we likely have due to taking the mean of all features in our playlists, and the Bayesian multi-logit regression assumes linearity of independent variables and log odds, which is likely not the case in this classification problem.

The most constraining factor on our predictions, I believe, is the assumption that the mean of all features in a playlist is an accurate representation of the genre of a playlist. Many playlists are not created as “genres” to begin with. For example, for the playlist called “pump”, should this be hip-hop, pop, rock, or edm? While a playlist might be the sum of its songs, taking the mean of all features has the capacity to be affected by outliers, and we simply aren’t using enough classes to get closer to the true genre of a given playlist. As a future task and potential improvement, it may be worth taking the median of features instead, as this is less prone to being affected by anomalous songs in playlists. Despite this, songs are diverse. A song can hardly be classified to a single genre, and in order to do so, Spotify now has over 5,000 genres (Davison, 2020). Hence, given I was only querying from 10 genres for my songs which I used to train my models, it is likely that we poorly classify playlists.

Despite not making perfect recommendations, the models are interpretable, implementable (through the abstracted functions I created), and successfully output results. Thinking about Euclidean distances in high dimensional spaces as predictors of similarity is not novel, but very few have implemented such an approach for Spotify song

recommendation, especially through using a blend of supervised and unsupervised algorithms.

Some improvements beyond testing the median as a better measure of a playlist's features, could be to use a sentiment analysis approach on the name of the playlist as well. If I could rank the name "pump" on a scale of 0 to 1 in terms of low to high energy, for example, then we would have another feature to predict on. I could also change methods entirely. As seen in past challenges with Spotify datasets (Hamed Zamani, 2019), most high-performing teams use collaborative filtering where they "create an incomplete playlist-track matrix and use matrix factorization to learn a low-dimensional dense representation for each playlist and track. They learn similar representations for the tracks that often occur together in user-created playlists."

Further extensions of my research could be to visualise the 'tanh' layer in my neural network as an embedding layer in a 3D space. This would allow further insight into the performance of my neural network so that I can fine-tune it more and understand where it is struggling to classify. Also, if we wanted to compare my models further, we could design more appropriate statistics to analyse uncertainty.

In summary, I realise that there are a few fundamental flaws with my approach as highlighted, however, the results are somewhat promising and provide insight into a variety of machine learning and Bayesian methods for solving a relatively high-dimensional predictive inference problem. There are distinct improvements that can be made on my approach, such as having more computational power to extract larger samples and trying to use the median instead of the mean in playlist summarisation, and there is a lot of potential to apply more complex machine learning methods as well to this problem, as Spotify likely does to build its recommendation algorithm that is queried millions of times a day around the world.

References

Brownlee, J. (2016). *Dropout Regularization in Deep Learning Models With Keras*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Brownlee, J. (2019). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Davison, C. (2020). *Spotify Users Are Noticing Something Very Strange About Their Top Genres*. Retrieved from PureWow: https://www.purewow.com/entertainment/spotify-wrapped-genres

Gelman, A. J.-S. (2008). *A WEAKLY INFORMATIVE DEFAULT PRIOR DISTRIBUTION FOR LOGISTIC AND OTHER REGRESSION MODELS*. Retrieved from ArXiv: https://arxiv.org/pdf/0901.4011.pdf

Hamed Zamani, M. S. (2019). *An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation*. Retrieved from ACM Digital Library: https://dl.acm.org/doi/abs/10.1145/3344257

Keras. (2020). *keras.io*. Retrieved from Drouput layer: https://keras.io/api/layers/regularization_layers/dropout/

Maklin, C. (2019). *Gaussian Mixture Models Clustering Algorithm Explained*. Retrieved from Medium: https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e#:~:text=Gaussian%20mixture%20models%20can%20be,of%20the%20bell%20shape%20curve.

Sean M. O'Brien, D. B. (n.d.). *Bayesian Multivariate Logistic Regression*. Retrieved from Duke Statistics: http://www2.stat.duke.edu/courses/Fall03/sta216/lecture10.pdf

Spotify. (2020). *Explore*. Retrieved from Spotify For Developers: https://developer.spotify.com/

Spotify. (2020). *Spotify Million Playlist Dataset Challenge*. Retrieved from AIcrowd: https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

Stan. (2020). *Multi-Logit Regression*. Retrieved from Stan User's Guide: https://mc-stan.org/docs/2_25/stan-users-guide/multi-logit-section.html

Wikipedia. (2020). *Expectation–maximization algorithm*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm

Appendices

Link to Gist containing code:

https://gist.github.com/Briiick/9021a1a91c27a14d40c932ac68d07ae1

Additional predictions with:

Neural Network

```
Name of playlist: Berlini
The playlist is genre: jazz


The recommended songs, in no particular order, are:
Doubts 2, by Ibrahim Maalouf
I Wish I Played Guitar, by Liam Noble
Soar Away, by Dave Douglas
Birth, by Keith Jarrett
Winter, by Mark Isham
She Walked Into My Life, by Joe Albany
Ahmad's Blues, by Ahmad Jamal
Smoke Gets In Your Eyes, by Art Tatum
You Go To My Head, by Art Pepper
Blues Stride, by McCoy Tyner Trio
It Ain't Necessarily So, by Mary Lou Williams
Swedish Landscape - Live, by Chick Corea
Blackberry Winter, by Keith Jarrett
The Followers, by Pete Oxley
Rhapsody in Blue, by George Gershwin
Name of playlist: Guitar
The playlist is genre: jazz


The recommended songs, in no particular order, are:
Doubts 2, by Ibrahim Maalouf
I Wish I Played Guitar, by Liam Noble
Soar Away, by Dave Douglas
Birth, by Keith Jarrett
Winter, by Mark Isham
She Walked Into My Life, by Joe Albany
Ahmad's Blues, by Ahmad Jamal
Smoke Gets In Your Eyes, by Art Tatum
You Go To My Head, by Art Pepper
Blues Stride, by McCoy Tyner Trio
It Ain't Necessarily So, by Mary Lou Williams
Swedish Landscape - Live, by Chick Corea
Blackberry Winter, by Keith Jarrett
The Followers, by Pete Oxley
Rhapsody in Blue, by George Gershwin
Stardust, by Gerry Mulligan
Moonlight In Vermont, by Chet Baker
Black Orchid, by Cal Tjader
D-Natural Blues, by Wes Montgomery
I'll Let You Know, by David Hazeltine
Where or When, by Wynton Marsalis
He's Younger Than You Are - From "Alfie" Score, by Sonny Rollins
Requiem, by Lennie Tristano
My Funny Valentine, by Gerry Mulligan
```

K-Nearest Neighbour

```
Name of playlist: Your Top Songs 2020
The playlist is genre: rock


The recommended songs, in no particular order, are:
Go To War, by Nothing More
Jenny, by Nothing More
Alive - Chris Lord-Alge Mix, by P.O.D.
Semi-Charmed Life, by Third Eye Blind
Attack, by Thirty Seconds To Mars
You're Gonna Go Far, Kid, by The Offspring
Hard Times, by Paramore
Bitch Came Back, by Theory of a Deadman
Should I Stay or Should I Go - Remastered, by The Clash
Times Like These, by Foo Fighters
Misery, by blink-182
Feels Like Summer, by Weezer
Otherside, by Red Hot Chili Peppers
Who Do You Trust?, by Papa Roach
Know Your Enemy, by Rage Against The Machine
Name of playlist: My Chemical Romance
The playlist is genre: alternative


The recommended songs, in no particular order, are:
Machinehead, by Bush
Decode - Twilight Soundtrack Version, by Paramore
When The Sun Goes Down, by Arctic Monkeys
Go With The Flow, by Queens of the Stone Age
Kings And Queens, by Thirty Seconds To Mars
Wolf Like Me, by TV On The Radio
I Can Make You Love Me, by British India
She's Long Gone, by The Black Keys
Moth Wings, by Pond
Vasoline, by Stone Temple Pilots
Knights of Cydonia, by Muse
Walk, by Foo Fighters
Iris, by The Goo Goo Dolls
I Wanna Get Better, by Bleachers
Supremacy, by Muse
Echoes, by Klaxons
```

Bayesian Multi-Logit Regression

```
Name of playlist: JAMS
The playlist is genre: rock


The recommended songs, in no particular order, are:
Barracuda, by Heart
Enter Sandman, by Metallica
Raining, by The Front Bottoms
Summer Of '69, by Bryan Adams
Cat Scratch Fever, by Ted Nugent
Sultans of Swing, by Dire Straits
Gimme All Your Lovin', by ZZ Top
Run To You, by Bryan Adams
High Hopes, by Panic! At The Disco
Up Down (feat. Florida Georgia Line), by Morgan Wallen
Sweet Child O' Mine, by Guns N' Roses
Highway Tune, by Greta Van Fleet
I Was Made For Lovin' You, by KISS
Bleed It Out, by Linkin Park
Pumped Up Kicks, by Foster The People
```

```
Name of playlist: GIRL POWER
The playlist is genre: pop


The recommended songs, in no particular order, are:
Fake Love, by Drake
Firework, by Katy Perry
Trap Queen, by Fetty Wap
Saved, by Khalid
Now and Later, by Sage The Gemini
Creep On Me, by GASHI
That's My Girl, by Fifth Harmony
Happy - From "Despicable Me 2", by Pharrell Williams
Look Alive (feat. Drake), by BlocBoy JB
Am I Wrong, by Nico & Vinz
No Promises (feat. Demi Lovato), by Cheat Codes
F*ck Up Some Commas, by Future
Call Me, by NEIKED
everytime, by Ariana Grande
Ruin My Life, by Zara Larsson
7/11, by Beyoncé
Roll In Peace (feat. XXXTENTACION), by Kodak Black
Woman Like Me (feat. Nicki Minaj), by Little Mix
Wavy (feat. Joe Moses), by Ty Dolla $ign
Either Way (feat. Joey Bada$$), by Snakehips
```