ORIE 4741 Final Project Report

Akshay Yadava (aay29), Ben Polson (bsp73)

Playlist Creator

Abstract:

Spotify has lots of publicly available data on their music, including artist(s), genre(s), audio features such as energy and acousticness, popularity, and release date. We want to leverage that data so that if someone has a certain vibe or idea for a playlist in mind, they can choose these metrics themselves and our algorithm will make a playlist. The user first gets to choose whether or not they want to prioritize more popular songs. If they don't prioritize popularity, we simply use k-means or k-nn to determine the playlist. If they do, we want to first weight the data based on the most important features for predicting popularity, then apply k-means or k-nn so that the chosen songs will be more popular.

Introduction

The motivation for this project stemmed from a common issue many people have in our current day and age: falling into the same trend of listening to the same music from the same types of playlists too often. Using our knowledge of machine learning and dealing with big messy data, the idea of this project was to create an application that a user may input specific preferences regarding their music taste and be given a generated playlist.

Section 1: Data Analysis

1.1 - Data Collection

All the data used in this project was pulled from the Spotify API. There is a package labeled "Spotipy" that grants users the correct functions to gather data from the open-source Spotify API.

We used the spotipy package in Python to get data from the API. The only way to look up a song on the API is with its ID, so in an effort to get a huge dataset of popular songs, we first found a list of the 500 most popular artists from Rolling Stone. We then looked up these artists on the API and retrieved the song IDs of every single song by these artists available on Spotify. We then used those IDs to get general information on the song, and incorporated Spotify's audio metrics on each song. We created a dataframe with this data for over 100,000 songs. Since all of the information was pulled directly from the Spotify API, less than 100 rows had missing values so we simply dropped them (we checked to make sure these were not well-known songs). We figured this dataset is too large for running many machine learning algorithms, and also many songs are very unpopular, even from top 500 artists. Therefore, we dropped songs with a popularity of less than 40 out of 100. We also created many-hot vectors for each genre of the artist whose song was in that row.

1.2 - Data Cleaning

Our initial dataset of around 50,000 rows by 367 columns. One issue encountered was the repeated occurrence of the same songs in the dataset as some artists had released multiple versions of the same song. With some manipulation of the dataset, these duplicate occurrences were removed with the most popular version of the song being kept in the dataset. Furthermore, the artists were encoded as one-hot vectors.

The various features present in the initial dataset are shown in **Table 1.1**.

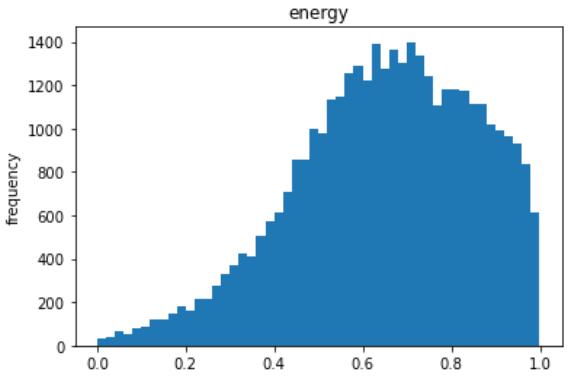
Table 1.1 Features Present in Dataset

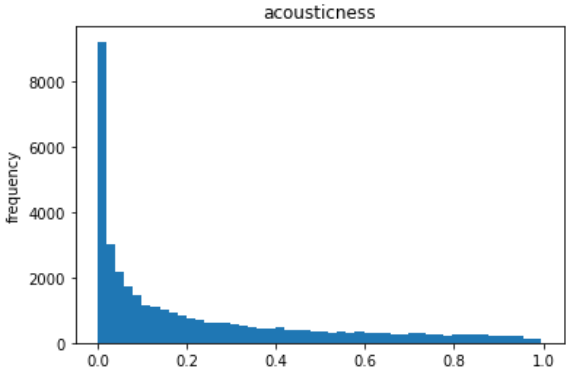
Continuous	release date, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo
Discrete	popularity, key, duration_ms
Boolean	explicit, features for each genre, and features for each artist

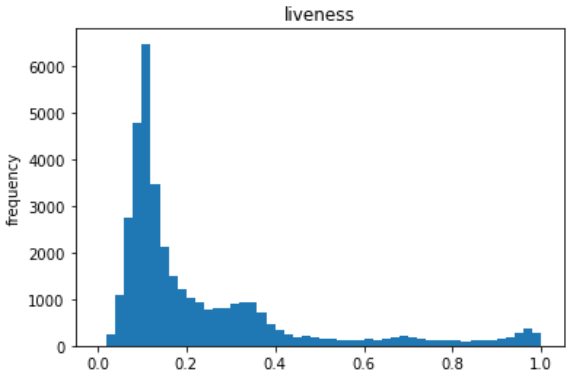
1.3 - Data Analysis

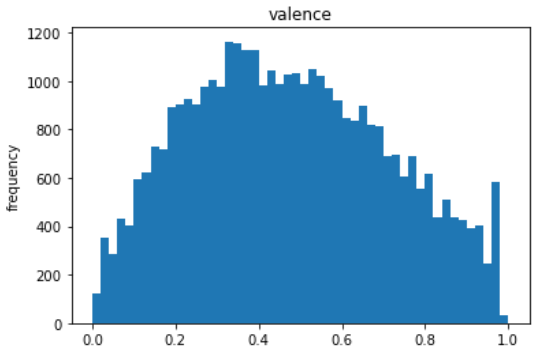
We initially plotted relevant histograms to see the distribution of various audio features in the songs. Some of these histograms are illustrated in **Figure 1.2**.

Figure 1.2 Initial Histograms Showing Audio Metric Distributions in the Dataset



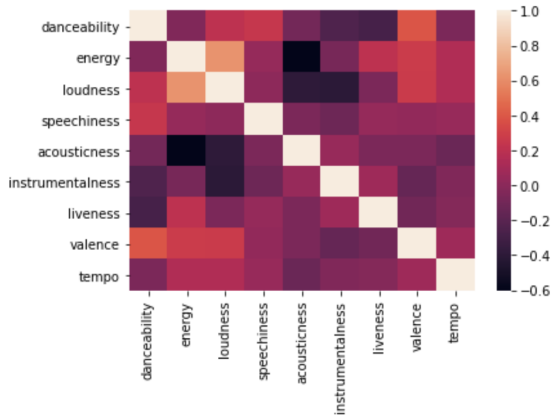






After reading the descriptions of each audio feature, the distributions of the histograms make intuitive sense. We also plotted a heatmap of the correlation between these features, shown in **Figure 1.3**.

Figure 1.3 Correlation between audio features



Since energy and loudness have a high positive correlation, and we aren't sure why the user would ever care about loudness (considering they can always change volume when listening), we decided to drop loudness when doing k-nn and k-means later on. Looking back, we should have trained the models in the following section without loudness, but it shouldn't make a large difference.

Section 2: Model Selection to Predict Popularity

2.1 - The Models Chosen and Why

The goal of this section is to come up with a sparse model that predicts popularity well. From there, we want to use the importance of each feature selected to rescale the data so that k-nn and k-means favor popular songs. The two sparse models we learned in class are lasso regression and control burn, so we tried both models and used the feature information from the model with low mean squared error and with a limited number of features selected to avoid the curse of dimensionality.

2.2 - Lasso Regression

Lasso Regression was run on the dataset in an attempt to try to predict the popularity of a song given its various features. The objective function for Lasso Regression is defined as follows:

$$\text{minimize} \quad \sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda \sum_{i=1}^{n} |w_i|$$

Lasso regressions minimizes the least-squares residuals between the expected and actual labels. In addition to this least-squares minimization, Lasso Regression implements L-1 regularization in the model to control the complexity of the solution. The use of L-1 regularization promotes simple, sparse models with few parameters. 5-fold cross-validation was used to find the most optimal regularization parameter lambda. The results of this test are shown in **Table 2.1**

Table 2.1 5-Fold Cross-Validation with Lasso Regression

λ	Avg train MSE	Avg test MSE	# Features Selected
0.0001	0.57707	0.62089	1089.8
0.001	0.57818	0.61906	979.2
0.01	0.63059	0.65107	422.2
0.01	0.84350	0.84392	7.0
1.0	1.00020	1.00025	0.0

From our results, we found that the most optimal lambda to be used in the L-1 regularization was 0.001. However, there were too many features selected for that lambda. Before trying to find a better value of lambda, we are going to try another model to see if it does better.

2.3 - Control Burn

Control burn attempts to grow a diverse forest of trees with a varied use of features to select the most important subset of features that represent the most important features used in predicting the label. In building this forest, we chose to simply bag trees rather than boosting then bagging or double bagging because bagging was the only method that worked with our large dataset. We are losing a little accuracy but our errors still ended up being lower than lasso regression. Cross-validation was also too slow, so we made a validation set out of 30% of the data and allocated the rest to training. After creating one model with an arbitrary lambda and getting lower error than lasso regression, we used telescopic search to find the best lambda based on both MSE and the number of features selected. A telescopic search varies the magnitude of lambda by a factor of 10 to find the best order of magnitude to be used for regularization. Once that order of magnitude is found, a more precise search is performed within that order of magnitude. Plotted in **Table 2.2** are the results of the first portion of the telescopic scope.

Table 2.2 Preliminary Telescopic Search with ControlBurn

λ	train_MSE	test_MSE	# Features Selected
0.01	0.07561	0.53351	10
0.001	0.07356	0.52261	555
0.0001	0.07307	0.52045	507
0.00001	0.07374	0.52178	547

From the initial portion of the test, we can see that around the order of magnitude of 0.0001, the mean squared error on the test set and the mean squared error of the train set are minimized. However, 0.001 has nearly the same MSE and it looks like the number of features selected converges at about 500-550. Given this information, another search was performed around lambda > 0.001 to get fewer features selected. The results of this second test are shown in **Table 2.3**.

Table 2.3 Second Telescopic Search with ControlBurn

λ	train_MSE	test_MSE	# Features Selected
0.002	0.07362	0.53857	424
0.003	0.07490	0.54942	162
0.004	0.07649	0.55584	83
0.005	0.07656	0.55993	74
0.006	0.07644	0.55317	36
0.007	0.07758	0.56388	27
0.008	0.07717	0.56064	27

From our secondary search, the number of features selected drops significantly between 0.005, and 0.006 with error also decreasing. We, therefore, chose lambda = 0.006 for our final model which was trained on the entire dataset (we have no need for a test set since we validated the model and we aren't generalizing to other

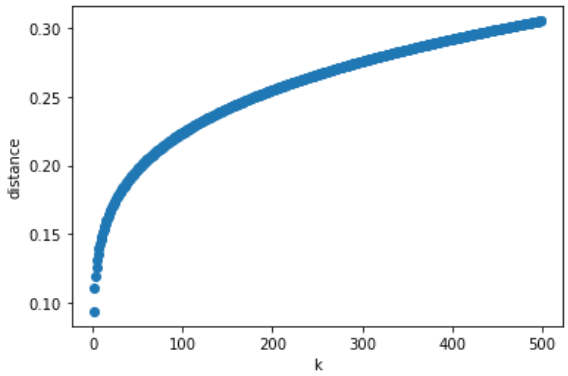
songs). 37 features were selected, and with their importances, we eventually created a rescaled copy of the data. To do this, we first fitted a least-squares model predicting popularity with just the 37 selected features, and we deemed positive coefficients as features with positive effects and vice versa. Next, we multiplied the positive features by e^(-feature importance) and the negative features by e^(feature importance) so the positive features are scaled-down and vice versa. The one part of this project we have yet to complete is using this rescaled data for k-nn and k-means in an effective way.

Section 3: Song Selection and App Development

3.1 - K-Nearest Neighbors (K-NN)

The first method used in our model of generating a playlist for a given user was the k-nearest neighbors algorithm. The k-nearest neighbors algorithm is a supervised learning problem that can be used to solve both regression and classification problems. Essentially, for a given test point, the distance (we used Euclidean) to all training points in the data is calculated, and the k neighbors with the lowest distances are the k-nearest neighbors. Once the k-nearest neighbors are found, the algorithm finds the most common label of the k-nearest neighbors for a classification problem or finds the mean of the k-nearest neighbors for a regressions problem. In our situation, we are using k-nn as an unsupervised algorithm where we are getting the name and artist of the k-nearest songs. One important aspect of k-nn is the choice of k. Therefore, to determine the most optimal k, the mean distance to the k-nearest neighbors over every point in the dataset were plotted (kind of like leave-one-out cross-validation). The results are shown in **Figure 3.1.**

Figure 3.1 Average Distance to K-Nearest Neighbors for All Points in Data



From this graph, it appears that the slope of the mean distance to the k-nearest neighbors begins to rapidly decrease from k=50 to k=100. We, therefore, used k=50 to generate the playlist. We chose 50 rather than 100 because 100 songs would be a very large playlist.

3.2 - K-Means

Another algorithm we tried using to generate a playlist based on audio preferences was k-means. K-means is an unsupervised learning algorithm that partitions the given dataset into k clusters. After cluster initialization (using k-means++), for every point in the data, the algorithm computes the distance to the nearest centroid (cluster center) of the k clusters and will assign that test point to that cluster. Once there are no new cluster assignments, the algorithm has converged.

Like k-nn, it is important to choose k so that the clusters are as accurate as possible. To choose k, the Elbow Method and Silhouette Method were used.

3.2.1 - Elbow Method

The Elbow Method involves plotting the sum of squared distances of samples to their closest centroid, shown in **Figure 3.2.** We choose k by finding the "elbow" in the graph. Since the elbow isn't that sharp, we also plotted the percent difference in the sum of squared distances from each k-10 to k to (**Figure 3.3**) For both of these graphs, we incremented k by 10 since training the clusters takes a long time.

Figure 3.2 Results of Elbow Method

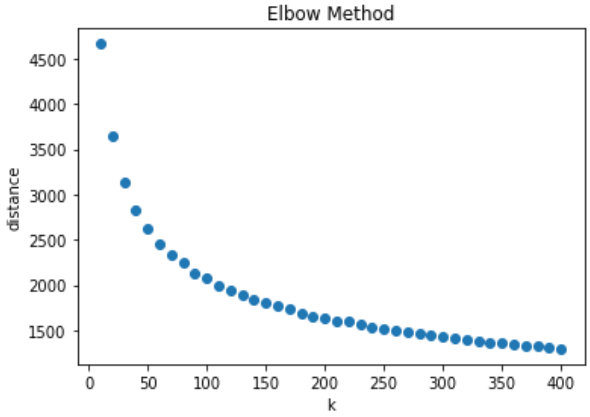
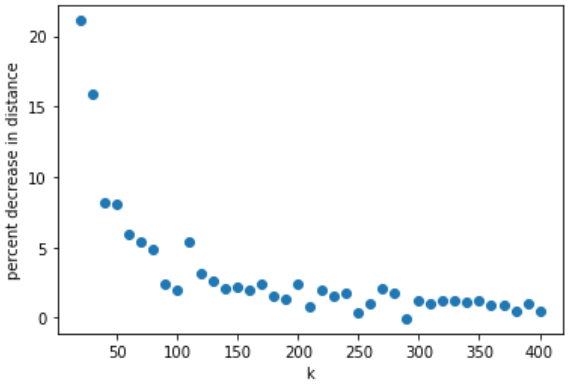


Figure 3.3 Percent Differences in Distance between each k - 10 and k



From these graphs, we can see that the sum of squared distances of samples decreases heavily between 0 and 100 clusters, and then decreases slightly from 100 clusters onwards. The percent differences take a big drop at k = 80, so we will use the Silhouette method for k = {75-85} inclusive.

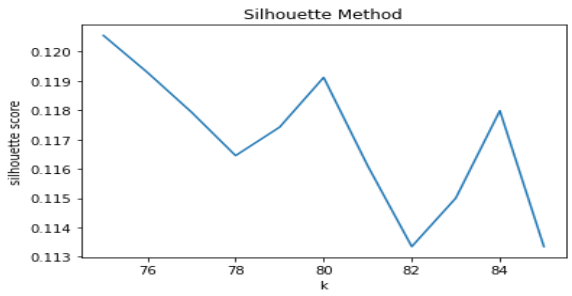
3.2.2 - Silhouette Method

The Silhouette Method is another method used to find the optimal number of clusters for an unsupervised learning problem. Essentially, the Silhouette Method computes the silhouette coefficients of each point and averages it out for all samples to get the silhouette score. The silhouette coefficient for a given point is defined as follows.

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)} & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1 & \text{if } a(i) > b(i) \end{cases}$$

In this notation, a(i) represents the average distance of the given point with all points in the same cluster, and b(i) represents the average distance of the given point with all the points in the closest cluster to its current cluster. The highest silhouette score is considered the best clustering. The silhouette scores are shown plotted in **Figure 3.4**.

Figure 3.4 Silhouette Scores vs. k



3.2.3 - K-NN with Centroids

Now that we know the number of clusters for the best clustering, the idea is to use k-nn with the centroids to find the closest cluster to the user's inputs. Once we find that closest cluster, we randomly select 50 songs from that cluster and that is the outputted playlist. Theoretically, the playlist from k-means will be a little more diverse than from k-nn since the songs we are selected aren't necessarily the most similar to the inputs.

3.3 - App Design

The app we designed is made up of python textboxes with prompts the user has to answer. The application allows the user to primarily decide if the popularity of a song should be a strong factor in creating a playlist for the user, or if they would prefer to provide general preferences and avoid popularity. The application then asks the user if there are specific genres or artists they do not like so that songs with those traits are dropped from the data. Furthermore, the application allows the user to input a range of years corresponding to what years the songs recommended were created, and also allows the user to enter a number between zero and ten corresponding to their liking of various common audio features in songs (danceability, energy, temp, etc). After cleaning the data based on genre, artist, and year preferences, it either uses k-nn or k-means based on a parameter of the app to find the 50 nearest songs. One thing to note about the app with k-means is that we can't look at graphs to choose k. Therefore, the first k to the nearest 10 where the percent difference is less than 5% is chosen for the elbow method, then the k with the max silhouette score between k-5 and k+5 is chosen as the final k.

Section 4: Ethics

4.1 - Weapon of Math Destruction

In lecture, a weapon of mass destruction was defined as a predictive model that has an outcome that is not easily measurable and whose predictions may have negative consequences that generate a self-defeating feedback loop. In our project, we utilized audio metrics of the most popular songs to generate a predictive model that generates a playlist of highly probable songs for a user to listen to based on their audio preferences. We do not believe this project creates a Weapon of Math destruction as a song's popularity is not necessarily hard to measure given popular songs have similar audio features. Furthermore, the predictions of our model could not harm any group as the playlist is solely a recommendation and incentive for the user to enjoy novel music. The only portion that may relate to some form of a weapon of math destruction is the idea that a song's popularity are purely subjective and vary based upon the given person, so a song's given popularity has no notion of being supported by fact.

4.2 - Fairness

Since the predictions of our model were made solely based on the audio features of the most popular songs and a

user's audio preferences, there are no features that are tied to protected groups and there is nothing protected by federal law. The predictions of the model are solely meant to introduce the user to new music and exhibit no bias in doing so.

Conclusion

We were most successfully able to predict popularity using control burn. We successfully used both k-nn and k-means to make a list of 50 suggested songs to a user based on their preferences about artists, genres, and audio metrics. The one thing we were not quite able to do was use the weighted selected features for predicting popularity to make suggestions that favor popular songs. However, we plan to continue work on this project in the future and implement this. Additionally, we realized that the version of duplicate songs where popularity is the highest is not necessarily the best choice because many older songs were re-released recently and those have the highest popularity. We will change the way we select songs out of duplicates by taking the oldest version and resetting its popularity to the popularity of the most popular version. Overall, we were able to apply many machine learning techniques learned in class and outside of class to create an app that can be very useful for people wanting to discover new music.