Michael Maaseide : maaseide.m@northeastern.edu
Jeffrey Krapf : krapf.j@northeastern.edu
Rishi Kamtam: kamtam.r@northeastern.edu
DS2500 Final Project Report
4/12/24

Final Project Report

Problem Statement and Background:

Music is played and cherished throughout the world and as a result of this, there is never any lack of suggestions for different songs that you should listen to. However, sharing music is a language and not one that is always easy for everyone to speak to others, especially people who just met or people from different cultures. This being said, sharing music is without a doubt a fantastic way of interacting and sharing who you are with other people that we feel everyone should be able to do. As our entire group has a strong interest in music and we have all experienced this problem, we wanted to look into this subject. Thus, we decided to use this project as an outlet to find a better way for people to give good song recommendations that have higher chances of their friends liking them.

As all of us in this group shared the common platform of Spotify we decided to do our project with this streaming platform as the primary method of giving recommendations. To start, we looked into how Spotify collects song data and learned that they characterize each song on the platform with features rated on different scales. This led us to the question of our project; do higher ratings in certain features correlate with more time spent listening to a song, and if this is true can we predict whether or not someone likes a song based on its characteristics?

Introduction to Our Data:

The initial data for our project comes from a JSON of Jeffrey Krapf's Spotify listening history. The JSON contains a list of dictionaries containing four keys: "endTime", "artistName", "trackName", and "msPlayed". The end-time keys contain a string corresponding to the time that the song stopped playing. The "artistName" key contains a string with the artist who recorded and published the song. The "trackName" key contains a string containing the names of the tracks performed. The "msPlayed" contains an integer containing the amount of msPlayed. For our project, we only pulled out the "msPlayed" and "trackName" data. **This data is also ethically sourced and consumed considering that**

Jeffrey Krapf consented to Spotify's data collection through my use of the app, and Jeffrey Krapf consented to the use of our data for this project.

After gathering the song names and creating a function to add up all the msPlayed, we used a get function to gather the Spotify ID. Considering our lack of experience using request functions, we sought help from online resources where we used a function created by Vlad Gheorghe which uses the name of a Spotify song to access the Spotify ID and then returns the id (a unique code) of the song as a string.

Immediately after finding the ID, the ID was used to gather the features for a track. The features were sourced from the Spotify API, using spotipy. This data is gathered ethically through Spotify's API**.** The features are metrics gathered through Spotify's web API reference that are used to describe a Spotify track numerically. These are the features that we collected: **Acousticness (**Float): "A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic." **Danceability (**Float): "Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable." **Duration_Ms** (int): The amount of milliseconds in a song. **Energy (**Float): "Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity." We received the energy of the track as a float. **Instrumentalism (**Float**)**: "Predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater the likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0" **Liveness (**Float): "Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live." **Loudness** (Float): "The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing the relative loudness of tracks. Values typically range between -60 and 0 db." **Speechiness (**float**)**: "Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words." **Valence (**float**):** "A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.

Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry)."

There may be implicit bias within the data. For one, the data doesn't take into consideration other avenues in which Jeffrey Krapf listens to music, so it's biased only to his Spotify data. Secondly, the features produced by Spotify may not give a faithful representation of the song.

Data Science Approaches:

Entering into the project we looked at the data and all of the characteristics were rated on fairly different scales that made the csv file difficult to understand at first glance. Both to combat this issue and make our data easier to visualize later on, we decided that we would begin by normalizing each of the features in our csv file. For each characteristic, we use min-max normalization and bring the rating for each feature to a scale of zero to one. Min-Max normalization is calculated using the equation: $\frac{x - min(x)}{max(x) - min(x)}$. After normalizing our features, we would dive into supervised machine learning algorithms that would lead us closer to answering our problem statement.

Getting into our regression, we would start by splitting our dataset into training and testing groups, using the train test split method. After doing this we will fit all of our data into a multiple linear regression model and compute the R^2 score. The R^2 score represents the proportion of the variance in the dependent variable explained by the independent variable. A higher coefficient of determination in the context of this project means that the data's relationship can be explained by a linear model. The R^2 score will give us a very general look at how the features correlated with the amount of time Jeff spent listening to a song.

After doing this, we decided that we wanted to look more into the specifics of how each of the features correlates with the time listening to a song. To accomplish this, we will retrieve all of the correlation coefficients, a metric that measures the strength of a linear relationship between two values, between each feature and the milliseconds spent listening. With the correlation coefficients (R values), we then wanted to visualize the results of our correlations. Thus, we decided to make linear regression plots

for the features that had the most extreme correlation coefficients. Finally, to see if we would get any better results, we plan to follow the same methodology but change the scope of the dataset to only songs that Jeff had listened to for more than ten minutes.

Our second approach to our problem statement was utilizing the supervised Machine Learning algorithm, K-Nearest Neighbors (KNN). The KNN classifier specifically addressed whether we can predict somebody would like a specific song based on its characteristics and ultimately could the classifier be used as a tool to recommend songs to others. The first step in building the KNN classifier was creating the classes for our data. We decided on two labels, "Yes" and "No". The "Yes" label was given to songs whose msPlayed was above the mean msPlayed and the "No" label was given to songs whose msPlayed was below the mean.

Our next step was finding the optimal K value. In this algorithm, K refers to the amount of nearest neighbors (pieces of data) that will be used to determine the class of a specific data point. Through research, we determined that the approach we wanted to take to find the optimal K value was using the square root rule, finding the square root of the number of data points we had. Data Scientist Amey Band outlines in an article, "The optimal K value usually found is the square root of N, where N is the total number of samples," supporting the methodology of our first step in building the classifier. Although we used the square root approach to finding the optimal K value, we decided that we wanted to test a range of K-values (+5 and -5 of the square root) to provide us with a more comprehensive understanding of how different K-values can impact the algorithm's metrics and performance.

We then decided to utilize K-fold cross-validation techniques to evaluate our KNN model with the range of K-values we had. To find the optimal K value within our range, we specifically wanted to measure mean accuracy a metric calculated using the equation: $\frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$ as we believed it would best fit what we were looking for in our classifier. We would then use the K-value with the highest mean accuracy as the K-value for our classifier.

As for the classifier itself, we would utilize the test split method instead of having a separate

testing data set. To measure the performance of our classifier, we decided to compute metrics such as accuracy, precision, recall, and the F1 score. The equation for accuracy is shown above, precision is calculated using the equation: $\frac{True\ Positives}{True\ Positives + False\ Negatives}$, recall is calculated using the equation: $\frac{True\ Positives}{True\ Positives + False\ Positives}$, and F1 is computed using: $2 * \frac{Precision\ x\ Recall}{Precision + Recall}$. The values of these metrics would give us a comprehensive understanding of how our classifier would perform when predicting ten specific songs and lead us one step closer to addressing our problem statement. Lastly, we would generate a heatmap for our classifier to get a visualization of how our KNN algorithm performed.

Results and Conclusions:

After normalizing all our features, we gathered the R^2 value for our multiple linear regression and computed a value of 0.01352. At first glance, we knew that this value was weak and did not support our hypothesis that higher features impact the time one spends listening to a song. However, we knew that our R^2 value for our multiple linear regression might not indicate a high linear relationship as multiple features can have contrasting effects on time played. Next looking at the individual correlation coefficients, our results are- Danceability: -0.0218, Energy: -0.0255, Key: -0.0131, Loudness: 0.0215, Mode: 0.0208, **Speechiness: -0.079**, Acousticness: 0.0137, Instrumentalness: -0.0497, **Liveness: 0.0118,** Valence: -0.0302, and Tempo: -0.013. After analyzing the individual R values, we concluded that the individual features also do not have a large correlation on msPlayed as all the features exhibited a weak positive or negative correlation.

To visualize our data, we decided to plot the features containing the weakest (liveness) and strongest (Speechiness) correlation coefficient values from our data set, represented by **Figures 1 and 2**.

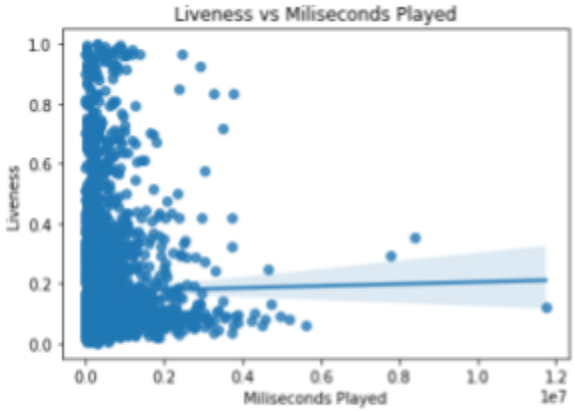

Figure 1: Liveness vs Ms Played

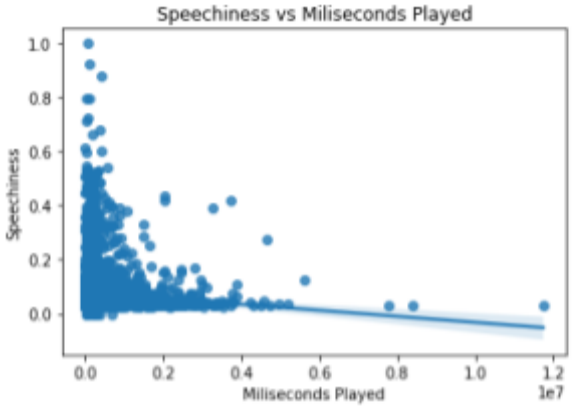

Figure 2: Speechiness vs Ms Played

Computing a poor R^2 value for our multiple linear regression and weak correlation coefficients for the individual regressions, we were hopeful that narrowing the scope to songs Jeff listened to for more than ten minutes would yield better results. Our R^2 value for the multiple linear regression of our new data set was 0.03847 more than doubled from before, but in the context of what the R^2 value shows, it meant almost nothing as it is still very weak. As for the individual correlation coefficient, our results were Danceability: -0.0413, Energy: -0.0339, Key: -0.0055, Loudness: 0.075, **Mode: -0.0013**, Speechiness: 0.0395, Acousticness: 0.012, **Instrumentalness: -0.1043**, Liveness: 0.033, Valence: -0.035, and Tempo: -0.0808. The feature with the strongest R-value was Instrumentalness and the weakest was Mode. Both of the relationships are exhibited in **Figures 3 and 4**.

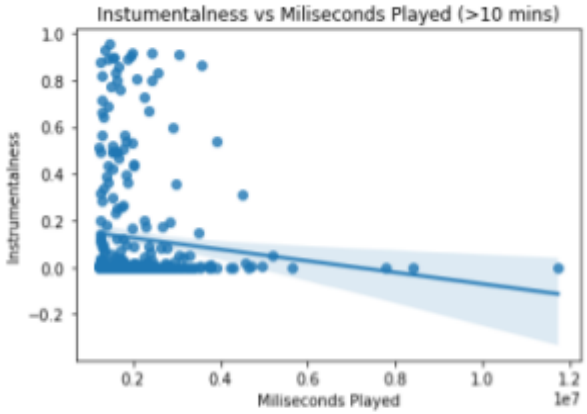

Figure 3: Instrumentalness vs Ms Played for Songs Jeff listened to >10 minutes

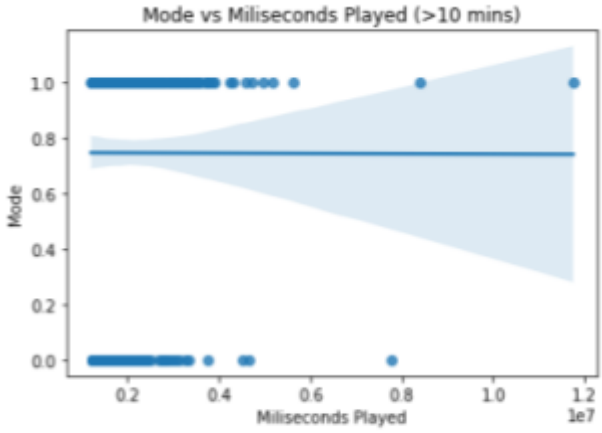

Figure 4: Mode vs Ms Played for songs Jeff listened to >10 minutes

Overall, the results of our multiple linear regression from the R^2 score to the individual R values proved to be poor and led us to the conclusion that there is no correlation between the features and the time played for Jeffs song data. Additionally, changing the scope unfortunately did not change our results, as the metrics still led us to the conclusion of a weak linear relationship.

As for our KNN classifier, the results of our methodology gave us a thorough insight into our problem statement and overall supported the results we gathered through the multiple linear regression. To start, we found the mean of msPlayed for our data set which was 433104.35 ms (Roughly 7.2 minutes) , and assigned labels based on where the songs fell (above or below the mean). We then calculated the square root of our 4169 data points and got 65. As we wanted to test a range of values +5 and -5 around the square root, we then used K-fold cross-validation to compute the mean accuracy score of the K-values between 60 and 70. Doing this, we found the most optimal to be 69 with a mean accuracy score of 71.57%. **Figure 5** supports our findings of 69 being the optimal K-value. While other K-Values seem to have similar performances, 69 emerged as the most optimal value when looking at the decimal values.

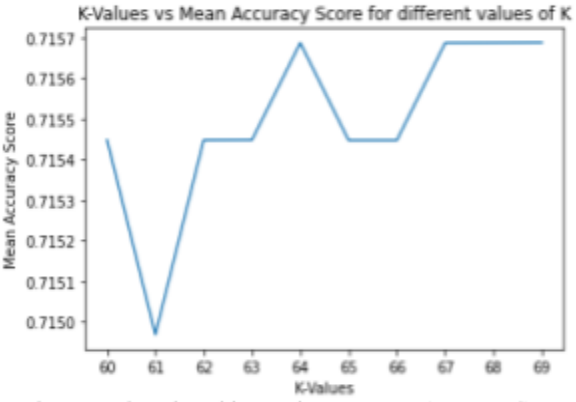

Figure 5: Line Plot with K-Values vs Mean Accuracy Score

With a K-Value, we then built our classifier, and using the train test split model we were able to gather the metrics, accuracy, precision, recall, and F1. The classifier's accuracy score was 73.896%, precision score was 100%, recall score was 0.366%, and F1 score was 0.73%. While our accuracy score proved to be decent, the perfect precision score and extremely low recall score provided us with the insight that our KNN model was poor at classifying positive instances and had a considerable number of false negatives. Additionally, our F1 score of 0.73% summarizes the trade-off between having an exceptionally high precision and a low recall score, showing us that our model performed unsatisfactory, specifically when classifying positive instances ("Yes" label).

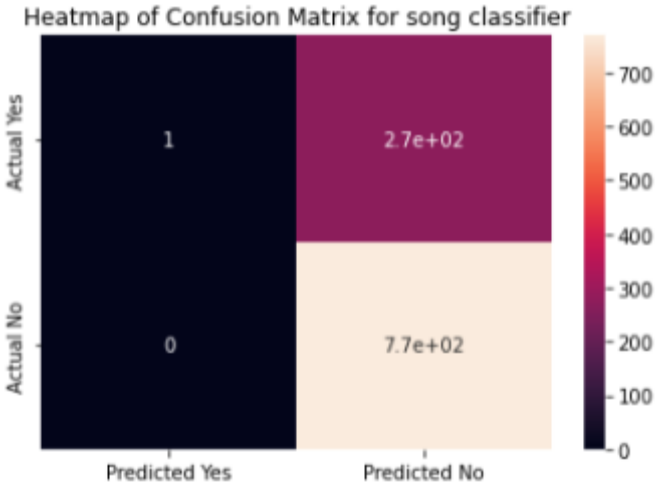

Figure 6: Heatmap of Confusion Matrix for Classifier

The heatmap shown in **Figure 6** supported the results garnered by the metrics as one can see how only one song was given the "Yes" label and was predicted "Yes" which led to a 100% precision. Additionally, the heatmap gave us context as to why the recall was extremely low as 270 songs were given the "Yes" label but were predicted "No." Overall, the heatmap

showed us that our predictor was good at classifying the ‘No” labels but subpar when identifying positive instances. Overall, the heatmap reinforced and gave an understanding of the metrics calculated through the KNN classifier and visualized the poor performance of the classifier.

With the classifier built and tested, the last part of our methodology was seeing if we could predict whether somebody would like a specific song. To do this, we tested ten different songs, seven songs not found in the data set, and three already existing songs in the data. Additionally, two out of the three already existing songs were given the “Yes” label. We decided to split the testing songs this way because we wanted to see if our classifier was at least getting the labels of the existing songs correct before moving on to new songs. However, all ten songs were given the “No” label which was not a surprise to us considering the poor statistics supported by the scores and heatmap.

Overall, the results of the multiple linear regression and the KNN classifier were sub-par and reject our hypothesis, that higher ratings in certain features correlate with time played. Additionally, the weak correlation between the two variables is undoubtedly a contributing factor to the inferior metrics generated by the classifier. In conclusion, through our methodology and results we can conclude that the features of songs do not correlate with time played and a KNN classifier cannot be built to predict whether somebody would like a song based on its features.

Future Work:

After much reflection, our group has come to realize that the main issue with our classifier is the scope of Jeff’s listening history. Different genres vary immensely in their features, and considering how diverse our testing sample is, we decided that it’d be a productive decision to change our scope to playlists. Playlists contain a much smaller sample size and are normally aligned upon a common feature. Our future classifier will take a playlist’s ID, calculate its features, find the highest correlation between features and minutes, and classify whether or not a song would fit in the playlist. We’ve also put consideration into changing some of the features that we have. Some of Spotify’s features aren’t accurate, so we’d like to include in the future, the age of a song, and its genre, and give more weight to whether or not someone enjoys a track by a specific artist.

Works Cited

"How to find the optimal value of K in KNN?" *Towards Data Science*, 23 May 2020, https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb. Accessed 11 April 2024.

"Min-Max Normalization Share." *64bitdragon*, https://learn.64bitdragon.com/articles/computer-science/data-processing/min-max-normalization. Accessed 11 April 2024.

"vlad-ds/spoty-records: Extract your Spotify streaming history and use the Spotify API to obtain song features." *GitHub*, https://github.com/vlad-ds/spoty-records. Accessed 11 April 2024.

"Web API Reference." *Web API Reference | Spotify for Developers*, https://developer.spotify.com/documentation/web-api/reference/get-audio-features. Accessed 11 April 2024.