Music Analysis Paper

Using Binary Logistic Regression In Music Data

Ethan Wood

2023-07-12

Introduction

Music is something that is uniquely human. It is an incredible tool for sharing feelings of joy and happiness, or depression and sadness. You can better understand a life of someone else's, enhance your current state or mood, or feel emotions that can be difficult to express with words. If I were to take a guess, something this complex and in tune with the human condition would be one of the most difficult things to model and analyze. Yet, brilliant people have figured out a way to turn music into numbers.

Using the data from Kaggle.com, I will analyze a dataset of songs that Spotify has quantified with certain characteristics. I will use this data to answer the two questions:

1. What is the most accurate model that predicts if someone will like a song?
2. How can this model be optimized to my advantage using the results of question 1?

Question 1: Usually, we find new music because a friend recommended it, we saw a post online about it, or maybe it was because an artist you enjoy released new music. It can be tricky sometimes to find new music. I personally have had 'music droughts' where I cannot seem to find something new that I enjoy. This grassroots way of discovering new music is what we have used for years, before streaming platforms such as Spotify became relevant. But now, we have transformed music into data, and have technology to aid us. Using a computer as our new music guru is the next logical step. Companies like Spotify and Apple Music have started to do this, with computer generated playlists and personal recommendations to the user. In the dataset I will explore, I see how Spotify has quantified songs by gauging how much singing or acousticness a song has. Each song has their own attributes, and it makes it much easier to mold these songs into something that we can understand (or rather, a computer can understand) a bit better.

Question 2: Once there is a model, what can be done to improve it, or utilize it better? Using the results from question 1, I will use statistical techniques to explore some of the data that the model results with to build a function that can help us. Rather than leave this analysis with just a model, we will use the statistics given to help us make an intelligent decision on how to use the model best. Doing this will result in a more customizable use of our model, and show me how to interpret this data better.

Lastly, this paper is written to showcase my coding skills as well as my writing and problem solving skills, so all of the graphics and data analysis were coded and created by myself without the use of AI of a chatbot such as ChatGPT. A RMD file will be made available to see this document and the code I used to create it. Feel free to contact me via the platform you found this paper on, or visit my Github account to view this and other projects at 'https://github.com/ewoodnc'.

Data

This data was retrieved from kaggle.com ('https://www.kaggle.com/datasets/geomack/spotifyclassification') from user George McIntire (@geomaci). He was able to gather this data using Spotify's API ('https://

developer.spotify.com/documentation/web-api'). Spotify's API allows internet users to create applications that work using data from Spotify and their songs. The dataset he uploaded contains 2,017 unique songs, with 17 variables. 13 of the variables are attributes such as *danceability* and *duration_ms* that describe the song in some way. Two of the variables are the song's title and artist, which I did not use for this data analysis. The final variable is *target*, and indicates whether or not the song was liked by George using binary.

The attributes my model in Question 1 focused on are: *acousticness*, *danceability*, *duration_ms*, *energy*, *instrumentalness*, *loudness*, *speechiness*, and *valence*.

-*Acousticness* is a variable from 0 to 1 that describes how acoustic a song might be, with 1 being high and 0 being low.

-*Danceability* is a variable from 0 to 1 that describes how danceable a song might be, with 1 being high and 0 being low. This is created using other features from the song, like tempo and the beat of the song.

-*Duration_ms* is the length in milliseconds.

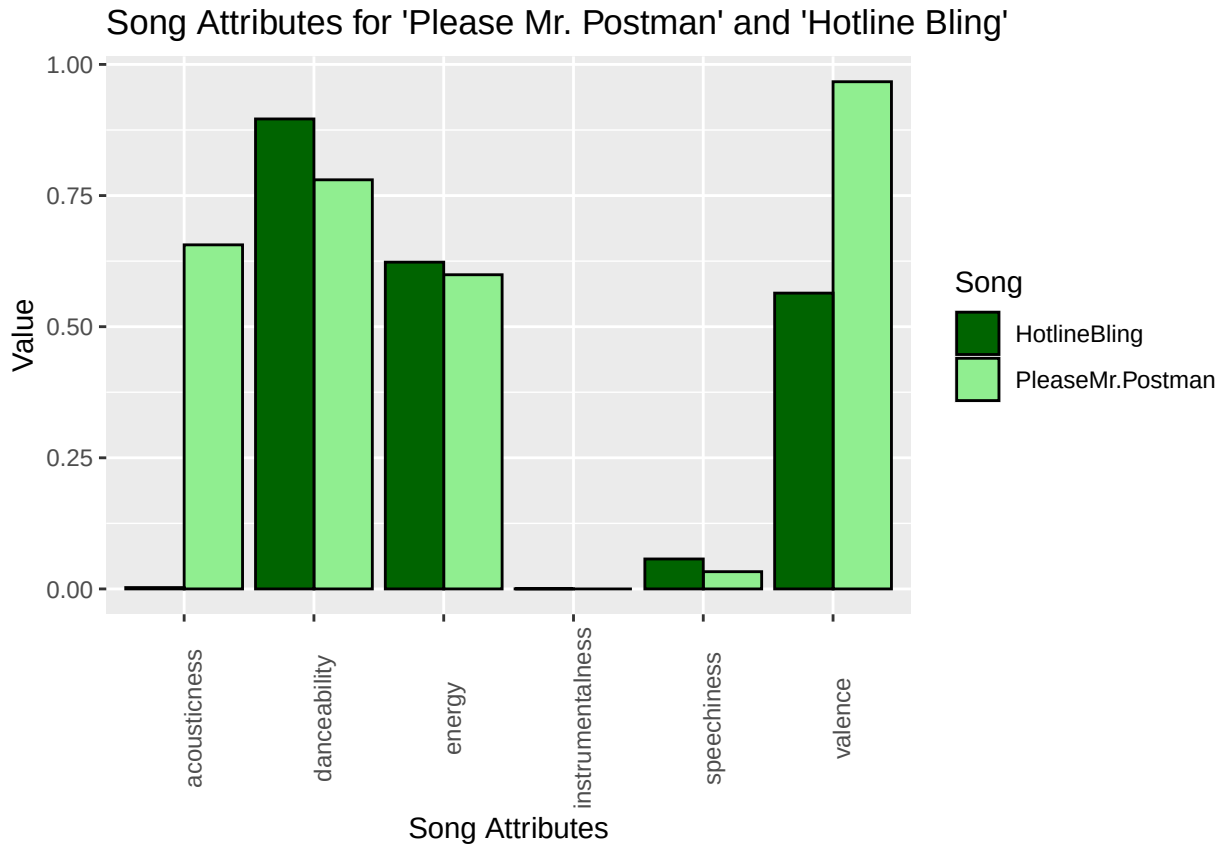
-*Energy* describes how much activity and intensity is in the song, with 1 being high and 0 being low.

-*Instrumentalness* describes how much singing or vocals are in the song, again from 0 being low to 1 being high.

-*Loudness* describes how loud the song is in dB.

-*Speechiness* describes the amount of spoken works in a song from 0 to 1. This is to help distinguish podcasts and audiobooks from music.

-*Valence* describes how happy or sad a song is. 0 is sad, and 1 is happy.



Here is a bar chart of 2 popular songs of a similar genre that vary in their release date. I only used song attributes that ranged from 0-1 to show the similarities and differences. 'Mr. Postman' is a Pop and R&B

song by The Marvelettes released in 1961. 'Hotline Bling' is similarly a Pop and R&B song by Drake, which was released in 2015. Mr. Postman scores a lot higher in *acousticness* than Hotline Bling does, which seems reasonable because Drake's hit is completely electronically produced. The *energy* of the two tracks seem to be about the same for both, which makes sense because each song is within the same genre. The *valence*, or measured "happiness" of the tracks vary. The lyrics show that Drake's song is about losing someone he loved, which is accompanied by a melancholy beat. Mr. Postman is about someone wanting to get a letter from someone they love, and as a listener, it feels like a happier song than Hotline Bling. Spotify's algorithm picks up on this and delivers Mr. Postman a higher *valence* value. *Instrumentalness* and *speechiness* are low for both songs because they are recognized to have vocals and instruments.

As a side note, this dataset is an example of why I find statistics amazing. The fact that people are able translate music into the world of numbers for people like myself to ponder over and analyze is amazing to me. Something that is not inherently quantitative based, able to be modeled and quantified. With limitations, of course. The ability to do this with things that are not very clearly quantitative, is why I study statistics.

Results

Question 1 The first question had the goal to find the most accurate model that predicts what songs George would like. My idea was to use logistic regression for predicting the binary response variable of *target*, which is the variable that says if he likes the song or not.

To start, I shuffled the data and separated 80% into a training group, and 20% of the data in a testing group. I used the **bestglm()** function to help me find a linear model that supplied the ln(*odds*), and inputted the training group data. Odds is sometimes refereed to as $\frac{\pi}{1-\pi}$, where π is the probability out of 1 of being a success, or a liked song in this case. The resulting formula was:

$$\ln(\frac{\pi}{1-\pi}) = -2.902 - 1.825x_a + 1.473x_d + 0.000002788x_t + 1.553x_i + 0.09367x_l + 4.330x_s + 0.7614x_v$$

Let the subscript a, d, t, i, l, s, and v represent the respective variables: *acousticness*, *danceability*, *duration_ms*, *instrumentalness*, *loudnes*, *speechiness*, and *valence*

I used a drop-in-deviance test to ensure that the model found by **bestglm()** is better than the full model with all possible variables. There was a drop of 11.3 on 7 degrees of freedom, which resulted in a p value of 0.1241. 0.1241 is greater than 0.05, so I fail to assume the full model is significantly better than the reduced model that **bestglm()** found.

To make the results easier to interpret, I used algebra to convert this linear model that results in $\ln(\frac{\pi}{1-\pi})$ to result in the probability π of George liking that specific song. Similarly, let b, a, d, t, i, l, s, and v represent the respective coefficients listed above.

$$\begin{aligned}
\ln(\frac{\pi}{1-\pi}) &= b + ax_a + dx_d + tx_t + ix_i + lx_l + sx_s + vx_v \\
\frac{\pi}{1-\pi} &= e^{b+ax_a+\ldots+vx_v} \\
\pi &= \frac{e^{b+ax_a+\ldots+vx_v}}{1+e^{b+ax_a+\ldots+vx_v}}
\end{aligned}$$

Now, there is model in terms of the variables and coefficients that **bestglm()** found that gives us the probability π of George liking a song or not. I decided that any song that returns π greater than 0.5 is a song that George likes. The model correctly predicted if George liked a song 68.07% of the time with the training data. Naturally, if someone were to randomly guess if George likes a song, they would be right 50% of the time due to the law of large numbers. Thus, the model is able to outperform random selection by 36.14%.

Song Title	Artist	Probability	Prediction	Target	Correct
Please Mr. Postman	The Marvelettes	**0.06817472**	0	1	**0**
Hotline Bling	Drake	**0.28590127**	0	1	**0**
Swimming Pools (Drank)	Kendrick Lamar	**0.52317509**	1	1	**1**
Uma Thurman	Fall Out Boy	**0.27234666**	0	0	**1**

Here, there are four songs displayed that the model analyzed. George liked Please Mr. Postman, Hotline Bling, and Swimming Pools (Drank), but the model was only able to guess that Swimming Pools (Drank) was a song he liked. George did not like Uma Thurman, and the table displays that the model correctly predicted this.

When this model is applied to the testing data using the same standard of π greater than 0.5 to be a song that George likes, then the model correctly predicted if George likes a song 63.12% of the time. This outperformed random selection by 26.24%

Now that I had the probability π and binary response values for each song in both the training and testing datasets, I could find the cross validation correlation for each dataset and thus the shrinkage. It is important to check that the shrinkage is small because a large shrinkage indicates that the model was over-fitted to the testing dataset, and is not an accurate representation of the pattern that I seek to describe. The cross validation between the probability and the binary response for the training dataset was 0.371, and the cross validation for the testing dataset was 0.328. Squared and subtracted (train^2 - test^2), I found a shrinkage of 0.03. This is small, which indicated that the cross validation squared did not change very much between the testing and the training data. However, the initial cross validation numbers were not very large to begin with. This is because I am applying this statistical test on a binary variable, the response if George liked the song or not, and a continuous variable, the probability that George likes the song or not, rather than two continuous variables. Thus, the correlation is low, which meant the low shrinkage may not be as relevant. This is evidence that there was not too much over-fitting within the data, but it should still be taken with a grain of salt. This is done to show that I know how to compute shrinkage and what it means, rather than it providing something inherently meaningful to this paper.

Results (%)	Liked	Not Liked	Total
Predicted Liked	29.46	13.12	42.58
Predicted Not Liked	23.76	33.66	57.42
Total	53.22	46.78	100.00

This table gives more information about the testing data. In the totals section, 53.22% of the songs were liked and 46.78% of the songs were not liked. The model predicted that George would like 42.58% of these songs, and would dislike 57.42% of them.

What is much more interesting is that I can compare the actual results to the predicted results to learn more about the errors in our data. 29.45% of all songs were correctly predicted to be liked, and 33.66% were correctly predicted to be not liked. 13.12% of the songs were incorrectly predicted as false-positives, or type one errors. They are songs that the model assumed George would have liked, but in reality he did not like. Similarly, 23.76% of the songs were false-negatives, or type two errors. These are songs that were predicted he would not like, but instead were liked.

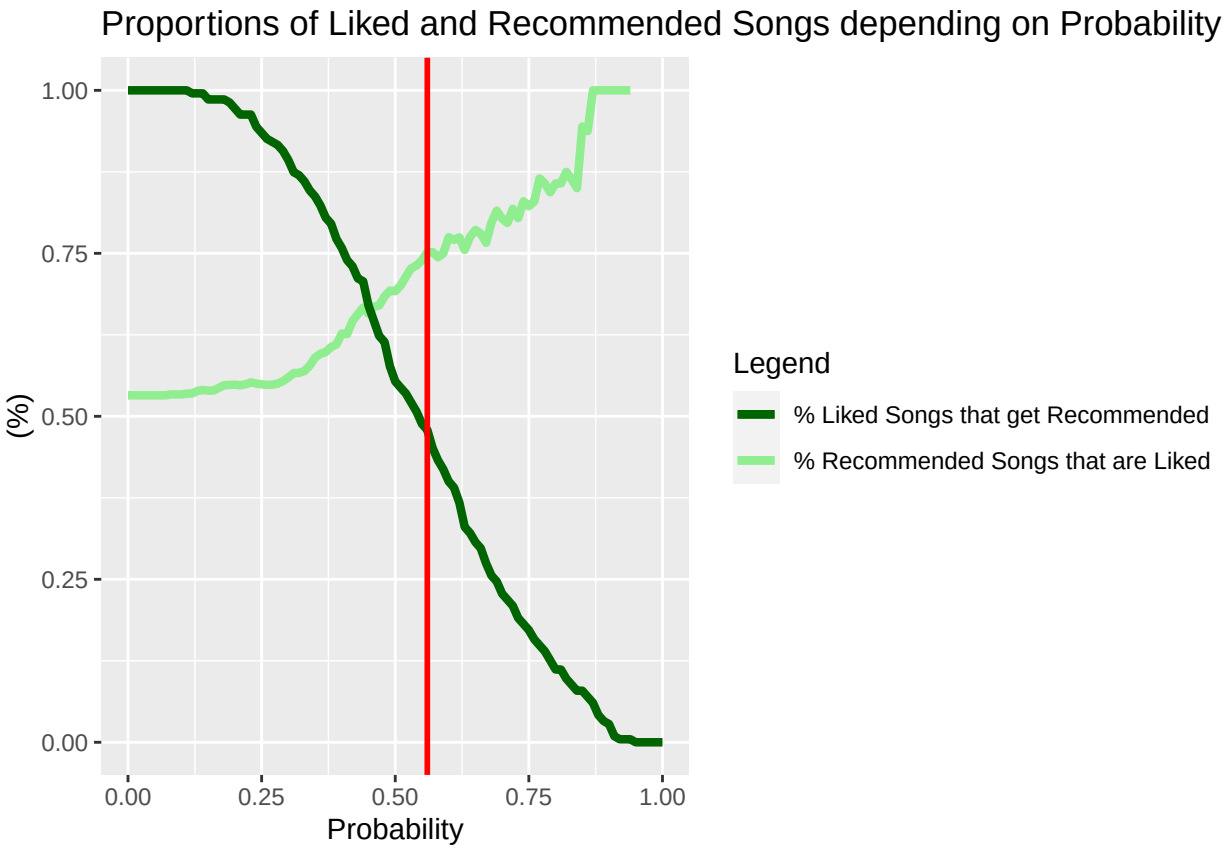
If I decrease the threshold of the probability π from 0.5 to 0.4, this is what that same table looks like now:

Results (%)	Liked	Not Liked	Total
Predicted Liked	40.35	24.01	64.36
Predicted Not Liked	12.87	22.77	35.64
Total	53.22	46.78	100.00

Naturally, the percent of predicted liked songs increased from 42.58% to 64.36%. I also saw the type one and type two errors change. The type one errors increased from 13.12% to 24.01%, so much more songs were predicted to be liked, when in fact they were not. This makes sense, because I lowered the standard of what was consider to be a 'liked song'. The type two errors decreased from 23.76% to 12.87%. So, decreasing the threshold of π decreases the number of songs that are incorrectly predicted to be not liked. So, the algorithm with a lower π will show more songs that would be liked, but at the cost of showing more songs that would be disliked.

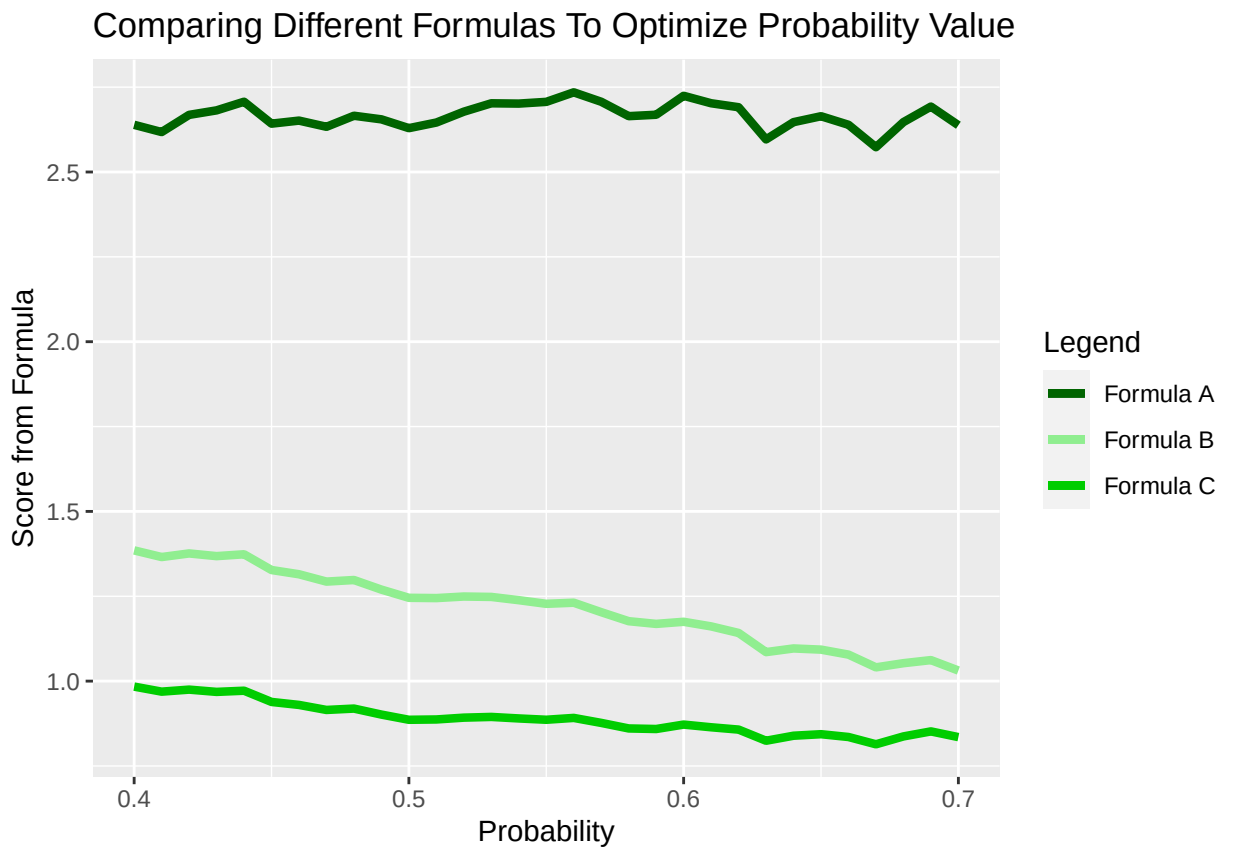
Specifically, when π is 0.5, 69.19% of the songs predicted that George would like are songs that he actually liked, but only 55.36% of all the songs I know that he would have liked are shown to him. The remaining 44.64% of songs he liked are not recommended to him, and he never gets to hear (at least from the algorithm). If π is changed to be 0.4, 62.69% of the songs the model recommended are songs that he actually likes, and 75.82% of all the songs he likes he will actually get to hear. This seems to be a much better value of π, because he only has to listen to a few more songs that he would not like to get the chance hear a lot more songs that he would like.

Question 2 The second question focused on how the model can optimized. In the last part of the question one results, I mentioned how changing the probability π of George liking a song changes both the percent of recommended songs that are liked, and the percent of all liked songs that are recommended. An ideal model would set π at a value that optimizes each of these values. In this graph, you can see how these percentages change as π changes.



Naturally, when the required π to be a liked song is low, all of the songs in the dataset will be recommended. So all of the songs George would like he gets to hear, but he also must hear all the songs that he will not like. When π is high, there were very few songs get recommended, but they are almost all songs that George would like. So what is the best value of π? A good place to start would be to decide how many songs someone is okay with hearing that they do not like. If three out of every four songs they would want to like, then the percent of recommended songs that are liked should be 75%. On the red line, when it is 75%, the percent of liked songs that will get recommended is a bit less than 50%, and the ideal π is 0.56. But is there a more concrete approach? While this is hands on, is there a formula someone can use to find this ideal value?

The answer is yes. However, to deal with the extremes of π I would set a max and minimum π value depending on your preferences. If at a minimum you wanted 25% of all of your liked songs to be recommended, and at the minimum to like 60% of all the songs that get recommended, then the range of π would be between 0.4 and 0.68.



Using that range of π, I created this graph showing how the weighted "score" of π values. Formula B was the most simple, and just adds the two %'s from the previous graph together. However, because the %'s do not vary equally there is a monotone decreasing relationship as π increases. The % of liked songs that get recommended decreases at a much faster rate than the % of recommended songs that are liked increases. This difference is what creates the decreasing functions. Similarly, formula C has a (almost) monotone decreasing relationship for the same reason. The formula used is squaring each %, adding together, then taking the square root. Because each formula is decreasing, the optimized π is 0.4 for each. Formula A values the % of recommended songs that are liked 3 times as much, then add this value to the normal % of liked songs that get recommended. This results in a relatively stable formula, where someone can retrieve an different optimized π of 0.56. The table for this π is:

Results (%)	**Liked**	**Not Liked**	**Total**
Predicted Liked	25.50	8.416	33.916
Predicted Not Liked	27.72	38.370	66.090
Total	53.22	46.786	100.000

Conclusion

This data analysis was done to explore my love for music and interest in data analysis. This paper focused on making a model to predict what variables from the Spotify API are the most useful in predicting if someone would like a song or not. Using the **bestglm()** function, I was able to find that *acousticness*, *danceability*, *duration_ms*, *instrumentalness*, *loudnes*, *speechiness*, and *valence* were the most important variables for this question. This model was able to accurately predict the data in the testing set 63.12% of the time, where the predicted probability π must have been at least 0.5 to be deemed a "liked" song. Analyzing the type one

and type two errors shows that around 7 out of 10 songs that the model recommends would be liked songs, but it would not recommend a little less than half of all of the songs George would like.

To me, this means the model is a bit underwhelming. If I were to use my personal Spotify data to recommend myself songs, I would be unsatisfied to know I am missing around one half of the songs that I might actually like. To solve this, I could lower π to be at least 0.4 (or lower) so I could see more songs that I would enjoy! If someone very picky only wanted to hear songs that would be almost guaranteed to be a liked song, they could set π to be very high.

Another reason this model seems to be a bit underwhelming is because of my lack of information about Georges listening habits. While the variables and how he was able to access the data from Spotify are very clear, I do not have the will power to go and listen to all 2017 songs in this dataset. Because of that, and a lack of a description on the Georges post, I am not entirely sure what kinds of songs are in this dataset. I assume and fairly confidently say that the liked songs in this dataset come from a mix of all the tracks that George has ever liked. From country and folk to death metal, the genres most likely drastically vary and thus so will the attributes. When the model sees two songs, one being a soft, acoustic, happy country song and a one being a loud, angry death metal song, and both equally being "liked", it confuses it. The attributes for these songs would sit at opposite ends of the spectrum, yet but would have a "1" in the target variable for being a liked song. So to compensate, the model must find some average of these songs, which in turn decreases the accuracy down to around 63%. Thus, I believe this model would perform better of a set of songs from a particular genre, rather than a diverse mixture of all of someone's liked songs.

For that reason, I believe that it would be worthwhile to test this model with a dataset of songs that all stem from a similar genre or theme. Then, I can see if it is more successful than the model I created today only by using a more specific dataset. If that were to work, then maybe this model would be better built for adding songs to playlists or building a catalog of a certain kind of music, rather than unspecifically finding more songs that you would like.

Another small thing to note is where the 'not liked' songs come from. Are they songs that George has listened to and claimed he has not liked? Are they an assortment of random songs from Spotify he put together? While the answer to this question does not affect the model per say, it will affect how I interpret this model and could explain its accuracy. This model was built assuming that the disliked songs are songs that are not pleasing to George's ears. From how the *target* variable is described, it is the correct way to interpret this. However, I believe the odds are low that he listened to over 1000 songs to compile this dataset. If they are simply songs George has not heard, that does not mean he will dislike them. If this model was made using songs that George might have liked as songs he disliked, then the integrity of this model crumbles. The model would not have an accurate sense of what songs he does not like.

The best (and safest) way to use this model might be to set π high, find a genre of music that you enjoy, then compile the largest list of songs you like and do not like from this genre. If then if possible, find a dataset of songs you have not heard that are from this genre. A random assortment of songs I believe would work too, but would not return as many songs you would like. Then, get your headphones ready, run the model, and get to listening!