Lazar Agoev, Stuart Ward, Zachary Trimmer
CS482-102
Professor Monogioudis
24 April 2022

Abstract

Introduction

The problem that we decided to tackle for our project was that of predicting popular music based on the trends that can be measured from previous years of music. Music is something that is most often considered to be very subjective, and many would argue that the popularity of a piece of music can be attributed to the whims of the audience that end up listening to said music. However, this may not entirely be the case, and that possibility alone is something that is worth investigating. The importance of looking for patterns in the world of popular music is because a bunch of people stand to gain from doing so. The world of music is inherently unpredictable, due to the subjectivity that was mentioned previously. Even with this sense of uncertainty, record labels and, by extension, artists can only put out so many songs each year. If there was some kind of method to determine how popular a piece of music was "expected" to be, it would allow for artists to tweak and refine their music in order to make it as popular as possible, and record labels could also use this data in order to manage their advertisement budget, putting more resources into the songs that are expected to be a hit with more people. Furthermore, discovering these possible patterns of what makes a popular piece of music is something that many data analysts, or just curious individuals in general, may take an interest in. This data could be used to make visualizations to show how certain aspects of a song, such as decibel sound level or music tempo, may impact how well it does with an audience.

After pursuing these questions and testing using our algorithm, overall we were able to find that, while our algorithm was able to slightly predict which songs may have been popular based off the dataset, this was also beset with some notable error and variance, as human emotion is something that is far too complicated for a machine learning algorithm to fully be able to predict.

Related Work

There have been many attempts to tackle the same challenge we have decided to pursue in this project here. Most notably, Columbia University's attempt fundamentally utilizes a dataset extracted from Echo Nest, chosen for its features and size. In this related work, there were a million songs each with its own h5 file. Since the dataset was too large, a randomly selected subset of 10,000 songs were used. There were a total of 41 features for each song used in this experiment like audio analysis, artist information, etc. Audio analysis featured factors like tempo, duration, mode, loudness, etc. It was noted in this work that many fields in the dataset were unusable due to old deprecated data. The many missing data fields resulted in them being dropped or the researchers using synthetic values in their place. The data exploration of this work was similar to what we were looking for. Feature importance, trends in our dataset, and identifying the optimal values for these features were some of the things that were common interests. The researchers in this work used many different models such as XGB, Logistic Regression, Random Forest, KNN, Decision Tree, and Support Vector Machines, whereas we only applied gradient boosting regressor. It should be noted that their best predictions came from XGBoost, which resulted in a AUC score of 0.68. Their overall conclusion deduced that it is much easier to determine if a song will be unpopular than to predict if it will be a hit, which we found interesting. The variables that can deduce if a song will be bad are much more evident and straightforward, whereas popular songs can have particular niches about them that may be hard

to predict. Moving forward, they plan to incorporate other features like artist location and release date to see if it will improve their overall model performance.

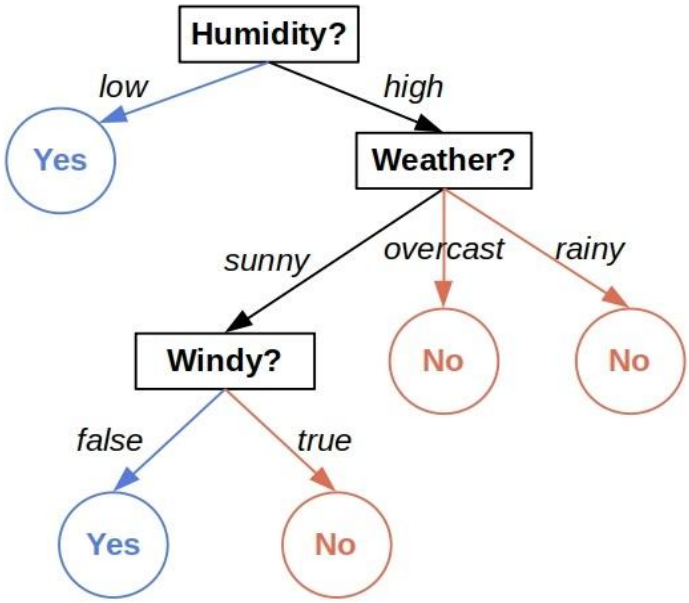
Data

The data that we used in order to facilitate our experiments was a mix of a few sources of data that we collected. The first two data sources were Spotify data that we found on Kaggle that detailed aspects of the top songs on the service for the years of 2019 and 2021, that we could use as a baseline in order to determine certain aspects of music that may end up making them more popular or appealing to audiences. This data included things like the BPM, energy of the song, decibel levels, the valence, the length, among a few other factors that are shown in the program and datasets themselves. In addition to using this data that was already available for download and use online, we were also able to find a website where we could input our own Spotify playlists, and the site would output the same types of factors for each song within that playlist or library that was inputted. Using this, we were able to collect a wider variety of data from various other genres in order to widen our dataset to test the algorithm on in order to try to tune it to the highest accuracy that we could. Overall, we had about 100 total samples of top songs from Spotify using the Kaggle datasets that were downloaded, and then a bit over 300 other songs that were taken from the online resource that allowed us to strip data from our own music, for a total of almost 500 songs that were placed into multiple data sets and loaded into the program using pandas. The data needed to be preprocessed slightly before it was usable with the algorithm. The data had to be trimmed in order to remove less relevant aspects such as the song's ID so that only the aspects that directly could influence the songs popularity remained to be used on the network.

In addition, we needed to collate the two popular song datasheets in order to train the algorithm on them at one time.

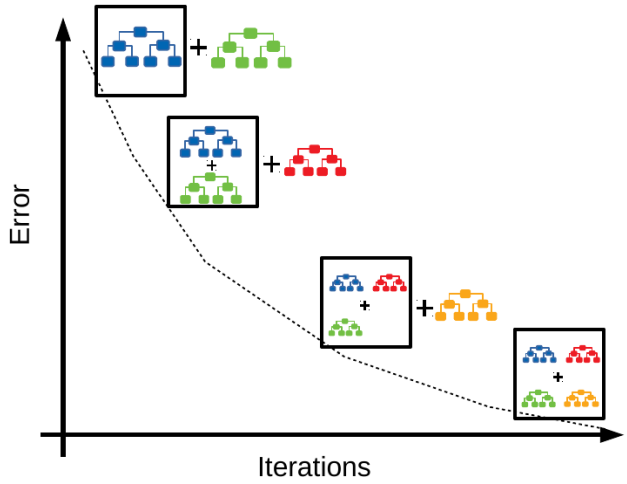
Methods

In order to solve the main problem that was outlined in the Introduction section above, we considered a couple methods that we thought would be best suited for getting the results we were aiming for with the data that we had gathered. We knew that we would be using some kind of machine learning in order to classify the songs we had into data that the algorithm could learn from, and then afterwards ideally we would be able to place any given song into the algorithm by recording the same metrics that we had trained the machine on (decibels, tempo, genre, etc), and that would then use our trained algorithm to output the expected popularity of that song. We believe that the best way to achieve these goals that we had set out was to use a decision tree algorithm. A decision tree is essentially a tree of nodes that an AI algorithm will create in order to inspect and check the features of an object that is input, and then travel down the tree to various subtrees until it reaches a conclusion. For example, the decision tree shown in the figure below is designed to see if it is a good day for playing outside:



With our project, we aimed to train a decision tree model in order to analyze the various elements of the songs that we input in order to come to a conclusion on how popular it "should" be. Instead of training just a singular tree, we of course aimed to train a whole grouping of trees in order to minimize random error, where this collection of trees is called a "forest". At first we simply considered using a Random Forest algorithm to build this decision tree, because we were enticed by how much simpler that algorithm was to tune than other widely used decision tree algorithms. However, there was also a notable drawback, that being that a Random Forest, while being overall quicker to run and easier to tune, was extremely prone to repeating the same types of errors over and over, since each tree would be constructed independently of one another, not allowing the system to learn from itself as well as we aimed for it to. The ease of this method was not worth the drop in accuracy, so we decided to seek a better method, which led us to what we eventually decided to use, which was Gradient Boosting. Gradient Boosting is another algorithm method that can be used to train forests, but the difference between this and Random

Forest, is that Gradient Boosting trains the trees one stage at a time, each time constructing a new tree in order to correct the errors of the previous one. In our case, this involved adding or tweaking a regressor to the tree, and this process was repeated until we hit a maximum number of trees in the forest, or when the errors reached an acceptable level, as demonstrated in the figure here:



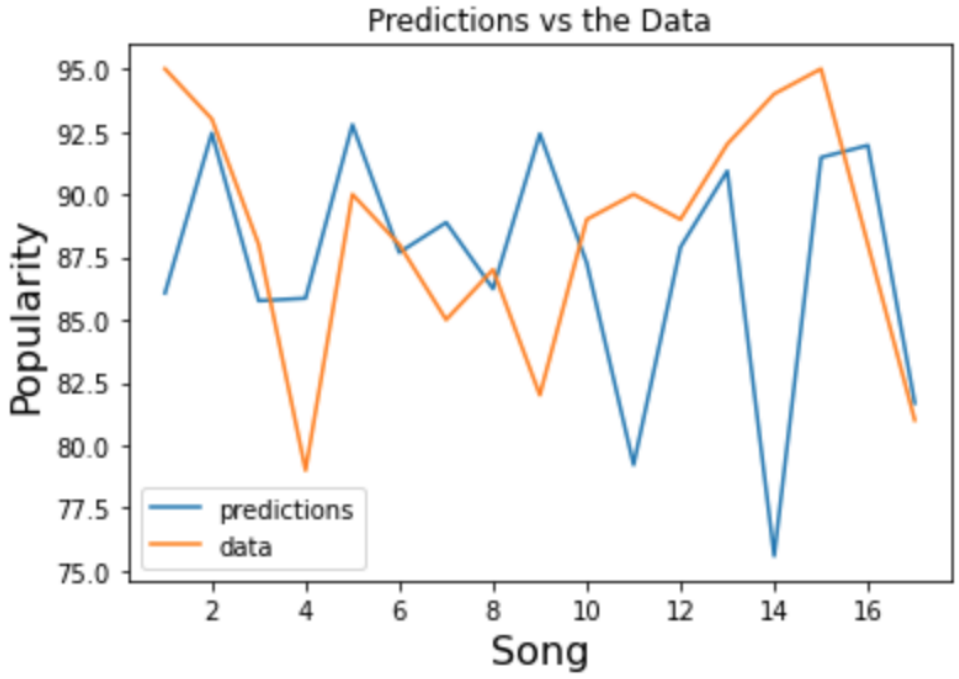
This was the final method that we settled on to train the data that we had, as the main drawback of Gradient Boosting is accidentally overfitting if you have too much data, but the amount of songs we had was actually lower than what we would have liked, so the risk of that was a lot more slim. This method was also the most conducive to being able to build off of itself based on multiple hyperparameters, which could be tweaked as needed to minimize the error as much as possible, and create the most accurate AI to predict the popularity of a song based on its various features.

Experiments

For our first attempt, our results were not as good as we anticipated them to be. With a mean squared error of 44.31, the visualization of the data vs predictions did not match the trends very well. We believe that the reason being that we did not utilize enough songs in our dataset, in

this case only 50. Furthermore, we deduced that because those 50 songs were from the most popular songs of 2021, there was a substantial influence of survivorship bias affecting the model. By only feeding the model popular songs, it fails to differentiate the features of that versus an unpopular song. The hyperparameters we used were as follows: n_estimator, max_depth, min_samples_split, learning_rate, and loss. We adjusted the values of the hyperparameters accordingly in order to obtain the best mean squared error value possible.

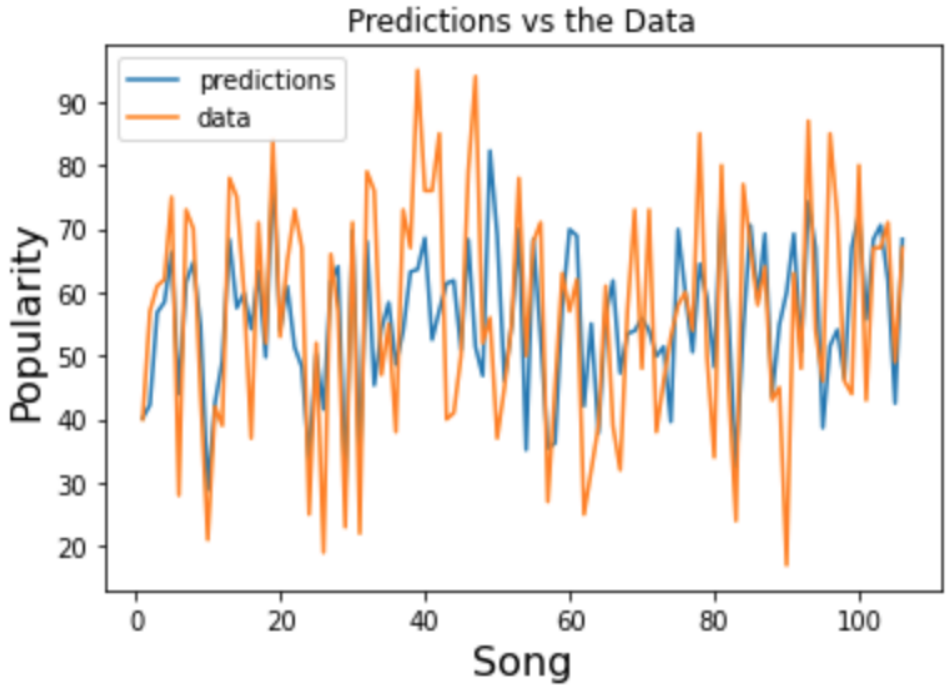
```
The Mean Squared Error is 44.31
```



In our second attempt, we found significantly better prediction results. We used the same hyperparameters, with values of n_estimator = 300, max_depth = 5, min_samples_split = 2, learning_rate = 0.01, and a mean squared error of 208.45. We chose these values based on a guess and check method, testing different numbers based on how the visualized graph responded. We found these results to be the best ones possible. It should be noted that we found this attempt more successful because the predicted values do follow the trends of the actual data. The mean

squared error increased from 32.95 to 208.45 due to the higher variance of the dataset since now we have more than just popular songs in it. Although we felt like we were on the right track, the predictability was not perfect. Even though we increased our dataset from 50 to 300 songs, it is still not high enough number to run a good optimization algorithm. Also, all of the songs have different genres which is a big variable that was not introduced into the algorithm. There were other common errors that we faced like others that have tried to do this. The toughest one being that at this stage human emotions are too complicated for a machine to understand how to predict what we will like. Although we can try to implement more features, it is hard to create a generalized approach towards predicting song popularity when everyone has their own unique taste of likes and dislikes.

```
The Mean Squared Error is 208.45
```



Conclusion

In conclusion, after all of the experiments were done, we were able to find out that during the first run through where we only used the top 50 songs from 2021, the algorithm did not do a very good job of predicting the popularity of the music. This is due to two main factors, the first being that only 50 data points is not a very high number to train on at all, ideally we would have a lot more to work with, which is why we added more data afterwards. Furthermore, since we only used data from 2021, this created a notable bias in the algorithm, as it was only able to predict popular trends in the past year, and did not have slightly less popular songs to train itself off of as well. When we did a second run through of the algorithm with more data that we had gathered, the algorithm was able to predict the trend in the music much more accurately, however there was still very noticeable error in the AI, as noted by the mean squared error that increased due to variance in the data. This error was likely borne from the fact that even increasing the dataset to 300+ songs is still not very high when it comes to machine learning, and also from us not being able to include genre into our algorithm, which is a big factor of what makes up music. If we wanted to improve this project in the future, we would try to collect a dataset of ideally around 10000 popular songs from over multiple years, and also would tune the machine learning algorithm so that it could take into account the genre of music as well. All in all, the main takeaway from our findings are that, while there are obviously some trends that are present in music that ends up popular, human emotion and bias are extremely subjective, and are far too complicated for an AI to accurately predict what songs will actually gain popularity simply based on a few objective factors.

Sources

https://www.baeldung.com/cs/gradient-boosting-trees-vs-random-forests

https://medium.com/swlh/gradient-boosting-trees-for-classification-a-beginners-guide-596b594a14ea

https://towardsdatascience.com/song-popularity-predictor-1ef69735e380