Spotify Song Success Prediction

Amelia Li & Rebecca Qiu

People sometimes struggle to find the right words when they try to explain why they love a song so much. “I don’t know, it’s just so good” can often be heard. There seems to be a missing layer between the language of words and the language of music. To translate the musical language into something more colloquial, Spotify started to quantify music at scale to see whether they can tune their algorithm to produce more popular songs, and hence generate more revenue.

Inspired by Spotify’s business strategy, we want to understand the interplay between data and music better. The essential question we are trying to ask is: “Can we use a song’s attributes to predict its popularity? If so, what attributes play the most important role?”

1. Data Overview

1.1 Data Description

The dataset utilized in this project is a Spotify Music Database obtained from Kaggle. The dataset is populated using Spotify API containing information regarding song genre, artist name, and track name, as well as audio features such as popularity, acousticness, danceability, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time signature, valence, and duration. The Spotify API reference goes into more detail as to how each audio feature is calculated.

The granularity of the dataset is based on individual songs with unique Spotify track IDs. The initial dataset contains 232,725 rows representing songs from 26 different genres. There are no missing values in this dataset.

1.2 Data Cleaning

We noticed that the dataset contains duplicate Spotify track IDs despite track ID being a unique value. The author did not address this, but we found duplicate rows containing the same values for a song except for the genre. We figured that having multiple genres for a song resulted in duplicate data, therefore we decided to remove duplicate rows by selecting the genre that appears first in the dataset for each song.

There was also a duplicate genre “Children’s Music” where two different apostrophes were used. This resulted in two of the same genres appearing in the dataset. We selected one of the apostrophes and replaced genres with the other apostrophe.

We converted the duration column that was originally in milliseconds to seconds, which is easier to interpret and thus more intuitive.

Time signature was originally a numerical feature on the Spotify API, but the author of the dataset changed it to a string. We decided to change it back into a numerical feature so that the number of beats in each bar can be utilized in our prediction.

Although the author of the dataset claims that there are around 10,000 songs per genre, we found that to not be the case after some data cleaning. The actual count ranges from around 100 to 10,000, with Comedy having the most number of rows and A Capella having the least number of rows. We are interested to see how this would affect the performance of our prediction, though we don't expect it to be a big issue due to the number of features we have.

Our final dataset contains 176,774 rows with each row having a unique Spotify track ID.

1.3 Dealing with Categorical Data

There are four categorical features in our dataset, namely time signature, genre, mode, and key.

We decided to Label Encode key since key is an ordinal variable and there is a form of ranking present that we want to preserve. The keys we have are C, C#, D, D#, E, F, F#, G, G#, A, A#, B, B and we encoded them from 0 to 11.

We decided to One-Hot Encode (OHE) the genre and mode variables since they are both nominal and we do not want them to be ranked if we use Label Encoding. We dropped the first OHE column for each variable to prevent multicollinearity.

1.4 Dealing with Outliers

According to the Spotify Web API, "time_signature" should be a value between 3/4 and 7/4 inclusive. However, there are "time_signature" values in our dataset with values 0 and 0.25, which could be a result of human error when the creator converted the time signature values to a string. We decided to drop these values as we were unable to interpret them. This reduced our data points by approximately 2300 songs.

2. Exploratory Data Analysis

2.1 Trend Descriptions for Quantitative Variables

2.1.1 Display distribution of popularity levels

We began exploring the data by visualizing the popularity distribution of the songs. Given the distribution of song popularity in figure 2.1.1, we saw that the majority of the songs cluster around medium popularity. Only approximately 8.5% of the songs in our cleaned data set have popularity values of 60 or above.

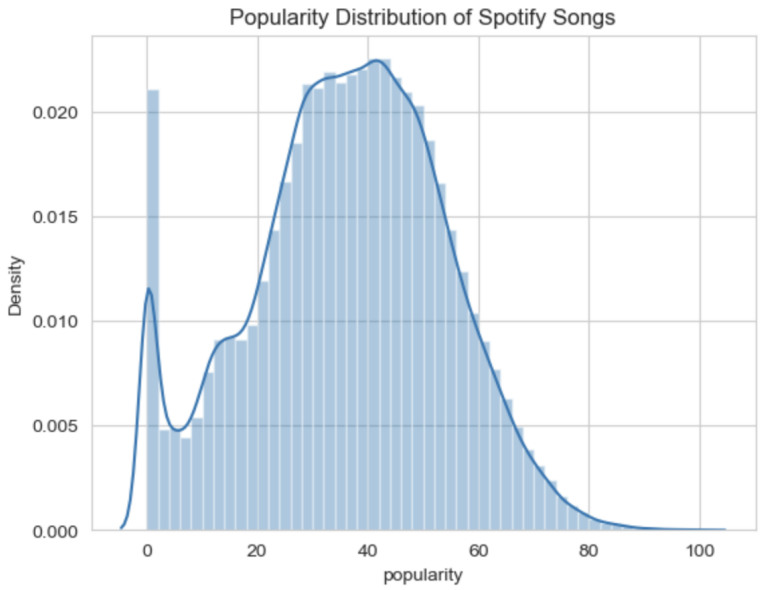

Figure 1. Density distribution of popularity scores

2.1.2 Modifying our target variable, "popularity"

We began by dividing the numerical variable “popularity” into 2 classes: “mega-hit” and “mega-flop”. The cutoff value for the 2 categories was the 50% percentile of the total number of songs in our dataset. The corresponding popularity score was 37.

The exact percentage of each class is shown in Figure 2.

	hit_song
mega-hit	51.537928
mega-flop	48.462072

Figure 2. Percentage of each song category of the binary classification model

Hence, we added a new column titled “hit_song” into our dataframe and assigned all songs with popularity above or equal to 37 to class “mega-hit” and the rest to class “mega-flop”. The newly defined variable “hit_song” would then become the target variable on which our models would predict.

2.1.3 Display correlations between numerical variables and popularity

Next, we pulled all the numerical variables into a new dataframe and plotted the heat map in Figure 2.1.2 to find if there exists any correlation between our numerical variables and song popularity. The numerical

variables are 'acousticness', 'danceability', 'energy', 'liveness', 'speechiness', 'instrumentalness', 'loudness', 'tempo', 'valence', and 'duration'.

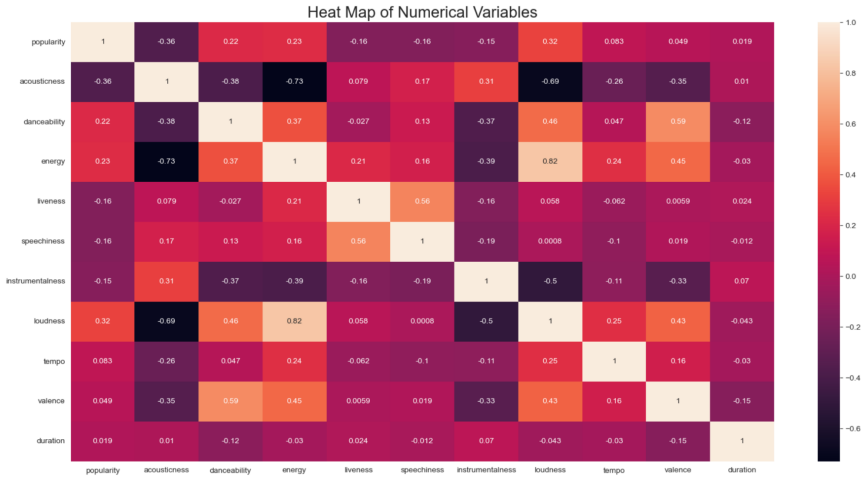

Figure 3. Heatmap of showing correlation between numerical variables

From the heat map we can see that features "energy", "danceability", and "loudness" have high positive correlation with song popularity while "acousticness", "liveness", "speechiness", and "instrumentalness" have high negative correlation with song popularity. Correlation of "tempo", "valence", and "duration" with song popularity seems to be close to 0. We can display the distribution of each feature against popularity to verify these correlation values.

2.1.4 "popularity" vs. "energy", "danceability", and "loudness"

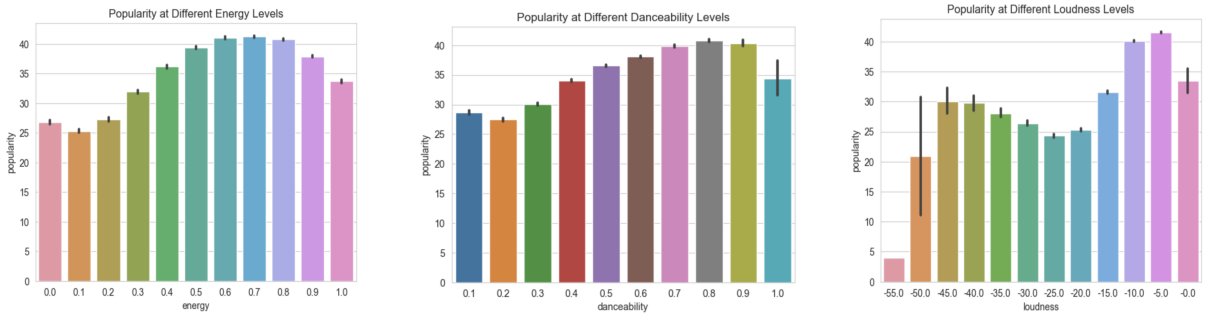

Figure 4. Popularity counts for various "energy", "danceability", and "loudness"

The bar plot for energy and danceability look very similar, with roughly the same peak in popularity at around levels of 0.7/0.8. The plot for loudness has the same general shape except for the lowest loudness level of around -55, where the level of popularity tanks to around 5. Since there is no visible confidence interval, we assume that this song is most likely an outlier and most songs will most likely not hit that level of loudness.

A quick Google search shows that the artist with the lowest loudness, Shakuhachi Sakano, releases deep meditative music for spas and sleep, which explains the loudness and the popularity.

2.1.5 "popularity" vs. "acousticness", "liveness", "speechiness", and "instrumentalness"

We can see that for these variables, the higher the value, the lower the average popularity level, which corresponds to the correlation values we obtained from the heat map.

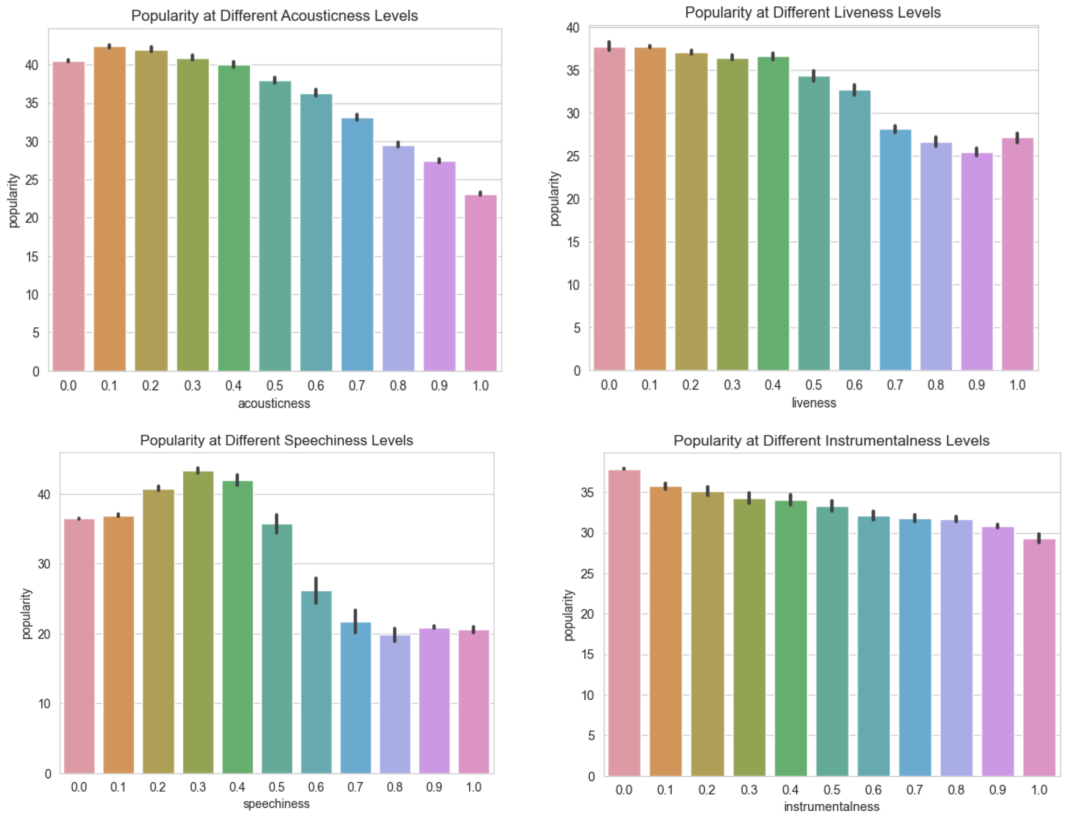

Figure 5. Popularity counts for various "acousticness", "liveness", "speechiness", and "instrumentalness"

2.1.6 "popularity" vs. "tempo", "valence", and "duration"

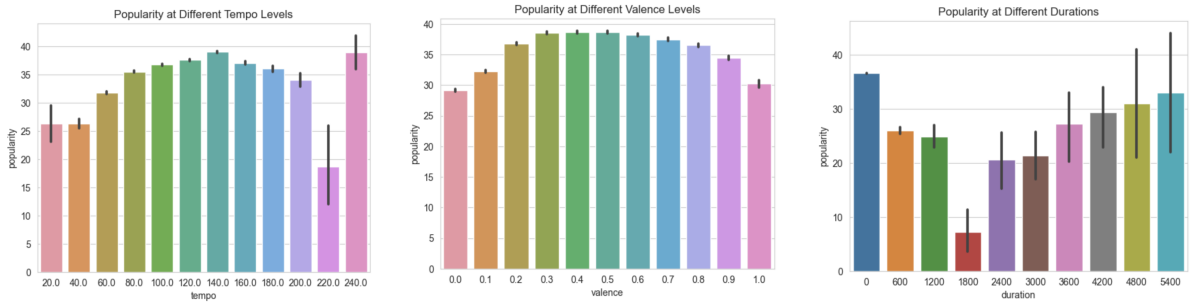

Figure 6. Popularity counts for various "tempo", "valence", and "duration"

We noticed that there are songs with durations of up to 90 minutes. These songs are mainly movie soundtracks or a compilation of songs. Those are not the style of songs we want to make our predictions for and will most likely skew our model. We decided to remove all songs that are longer than 20 minutes from our dataset and it further reduced the size of our dataset by 270 songs. We obtained the new distribution of popularity at different durations as shown in the graph below.

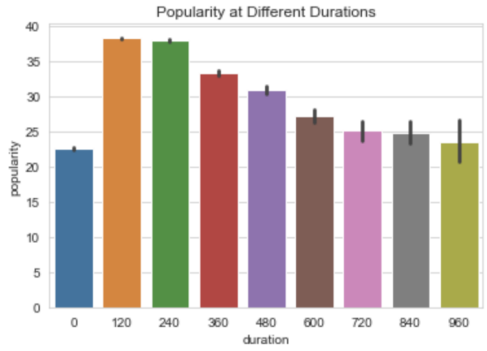

Figure 7. Popularity counts for truncated "duration"

2.1.7 Check multicollinearity using Variance Inflation Factor (VIF)

To determine the multicollinearity present amongst the numerical features in our data, we decided to use the Variable Inflation Factor (VIF), which determines the strength of the correlation between independent variables. VIF is calculated using the R^2 value, where $VIF = \frac{1}{1-R^2}$. At VIF = 1, there is little multicollinearity, when VIF < 5, there is moderate multicollinearity, and at VIF > 5 there is extreme multicollinearity that we want to avoid. We obtained the table of VIF values in Figure 2.1.3 for each numerical variable.There is extreme multicollinearity between many of our variables, with energy, danceability, and tempo exceeding 10. We will be using these values to guide us in feature selection when training our model.

	variables	VIF
3	energy	14.203668
2	danceability	13.728415
8	tempo	13.595389
7	loudness	8.704798
9	valence	7.100388
0	popularity	6.147280
1	acousticness	5.093762
10	duration	4.202035
4	liveness	3.358142
5	speechiness	2.375464
6	instrumentalness	1.913319

Figure 8. Sorted table of VIF values for each numerical variable

We kept removing features with VIF $\geq$ 5 by running the VIF function multiple times to obtain features with multicollinearity less than 5. We found that "liveness", "speechiness", "energy", "acousticness", and "instrumentalness" are not strongly correlated with each other, as shown in the table below.

	variables	VIF
2	liveness	3.187323
3	speechiness	2.115528
1	energy	2.114903
0	acousticness	2.055765
4	instrumentalness	1.478837

Figure 9. Sorted table of VIF values less than 5 for each numerical variable

2.2 Trend Descriptions for Categorical Variables

To find out whether categorical variables affect song popularity, we plotted the distribution and count of popularity scores for each categorical variable.

2.2.1 Popularity vs. "time signature"

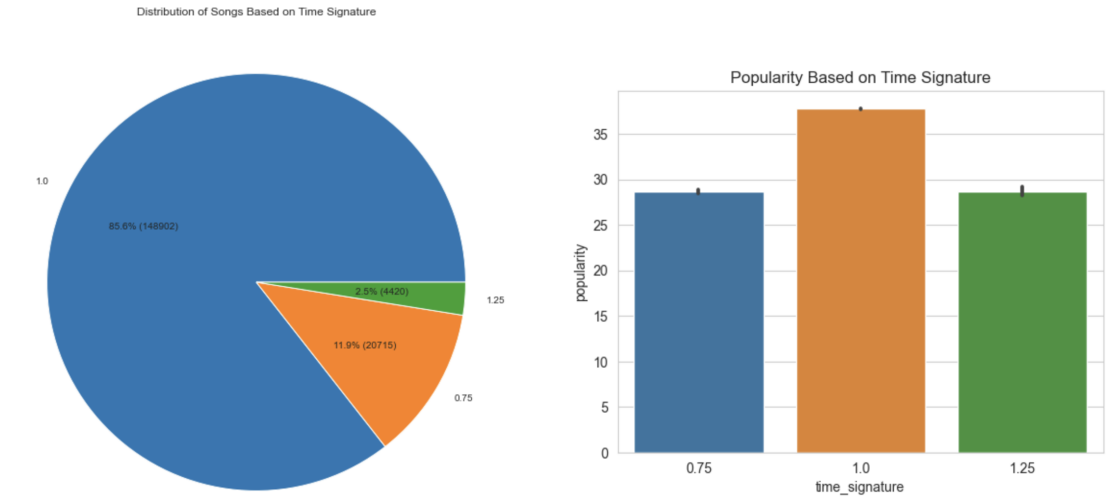

Figure 10. Popularity distribution and counts for "time signature"

2.2.2 Popularity vs. "key"

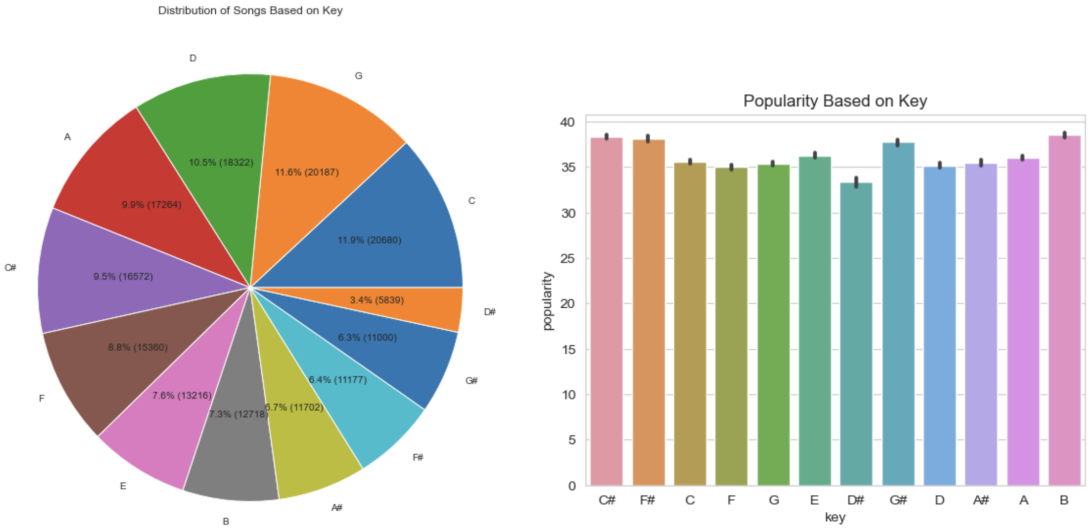

Figure 11. Popularity distribution and counts for "key"

2.2.3 Popularity vs. "mode"

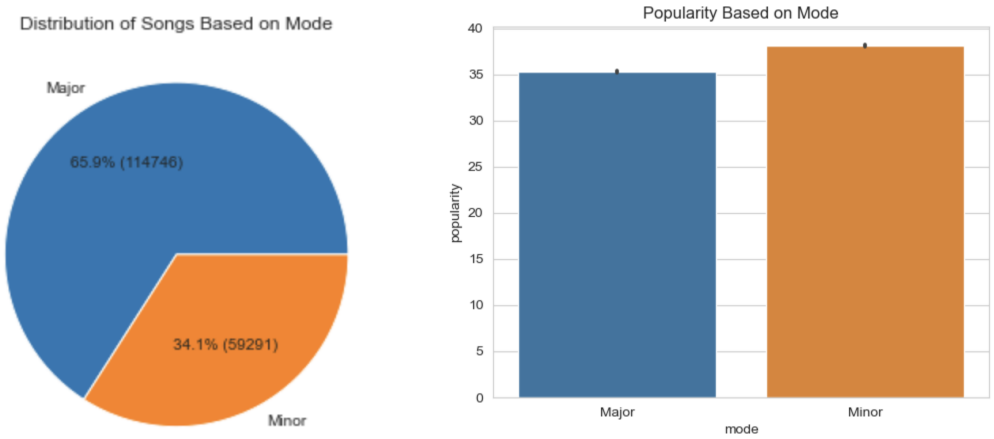

Figure 12. Popularity distribution and counts for "mode"

2.2.4 Popularity vs. "genre"

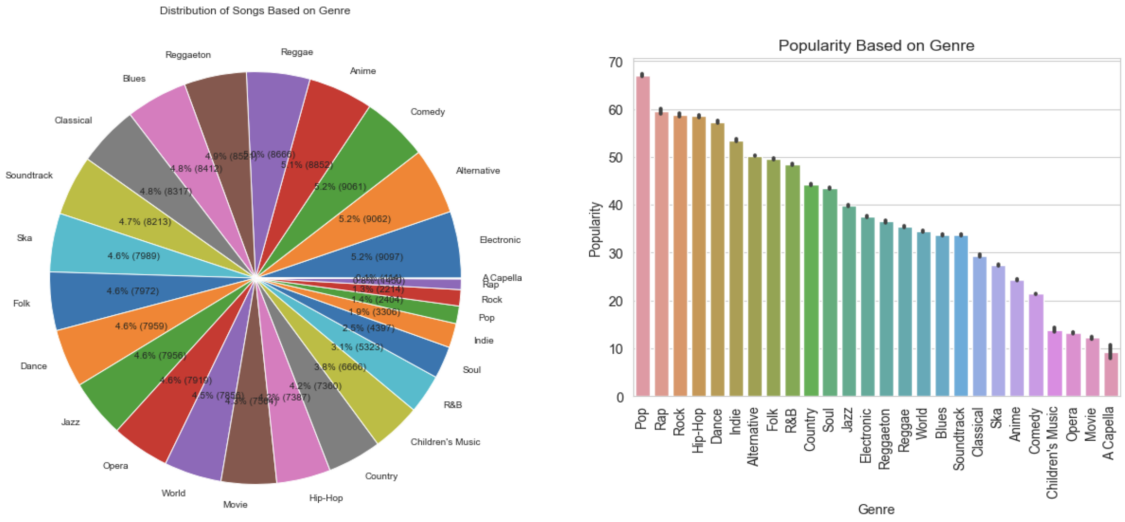

Figure 13. Popularity distribution and counts for "genre"

We saw the most significant variation in popularity across different music genres. Pop is the most popular genre while A Capella has a very niche market. Time signature also appears to be an important factor in deciding a song's popularity. On the other hand, the mode and the key of a song do not seem to play a major role here.

3. Project Question

Based on our initial findings and insights gained through EDA, we decided to revise our project question to the following: **Could we use a song's attributes to predict whether a track will become a mega-hit or a mega-flop**?

4. Baseline Model

4.1 Train-test-validation

All random states are set to 42.

To answer our project question, we constructed a baseline model using logistic regression.

First, we ran into the issue that our dataset is too big and training takes too long. Though it would be ideal to have a larger training set, we are unable to test the models that we would like to due to the time constraint. We decided to train our models with a much smaller subset of our data (10%) and a test set size of 5%. Our train size thus became 17,415 and test size was 8,621, which were both still sufficient.

Next, we further splitted the training data to obtain a validation set.

4.2 Logistic regression using all variables

To construct a baseline model, we fitted a logistic regression model to all of the numerical and categorical variables. We used multiclass One-VS.-Rest training scheme, 'lbfgs' for solver and 'l2' for penalty to train our model. The variable "hit_song" is our target variable. The train and validation accuracy scores are 0.8137 and 0.8194, respectively. We chose this to be our baseline model as it is not super trivial but relatively quick to train.

5. New Models

To see if we could make more accurate predictions based on song attributes, we performed hyperparameters tuning and features selection for the logistic regression model. We also constructed 4 more different models to see whether other models are able to perform better on our dataset.

5.1 Logistic regression with hyperparameters tuning and features selection

The hyperparameters we were looking to tune are the penalty term (L2 or no penalty) and the inverse regularization strength C for the logistic regression model. Through iterative tuning, we found the solver 'newton-cg' to perform the best. By using GridSearchCV to iteratively train the model, we found that C = 0.1 and penalty = 'none' gave the best performance with train and validation accuracy scores of 0.8183

and 0.8194, which was a slight improvement in training compared to the previous model but not in validation.

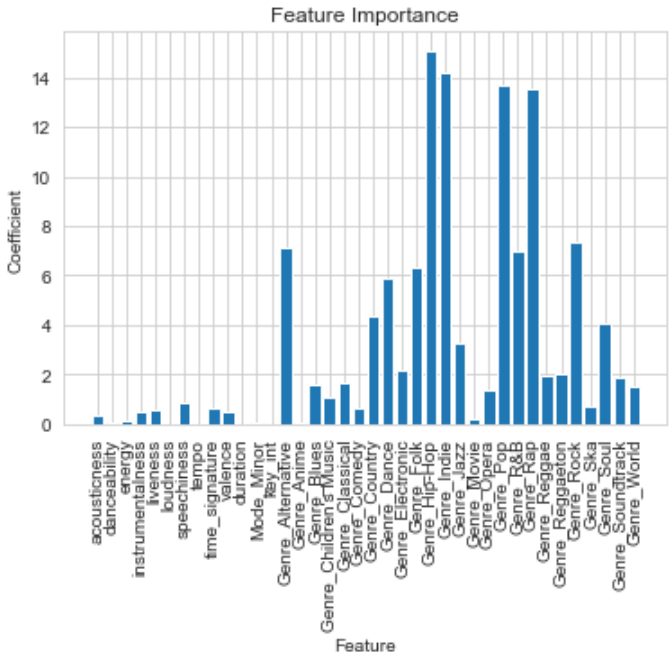

Figure 14. Importance of each feature for the logistic regression model

We noticed how the feature importance for the OHE genre column greatly surpasses that of the other features. We decided to train another logistic regression model with the same parameters using only the OHE genre features as we suspect that the other columns could be resulting in our model overfitting.

The resulting accuracies of the model were train accuracy = 0.8168 and validation accuracy = 0.8206. The slight drop in training accuracy indicates that our model was slightly overfitting. Dropping the columns resulted in a slightly better validation accuracy than before. We figured that the effort to perform feature selection vs. the results we were getting were not worth it. Hence we moved on to other models that we have learned in class to see how they perform.

5.2 Decision tree

5.2.1 Finding the best depth

Due to the number of features we have, we decided to begin with the decision tree model. We found the best depth of the tree to be 14 by comparing the validation accuracies given by tree depths ranging from 1 to 29. The decision tree of depth 14 produced train accuracy = 0.8779, and validation accuracy = 0.8079.

Figure 15. Train and validation accuracies at different tree depths

5.2.2 Bagging with decision tree of depth 14 as base model

Because the tree depth of 14 is quite large, we were concerned that our model might be overfitting the train data set. To decrease the variance, we chose the bagging method to make our model more robust.

With the decision tree of depth 14 as our base model, we used n_estimators = 100. The resulting train and validation accuracies were 0.9128 and 0.8217, respectively, which improved from the original decision tree.

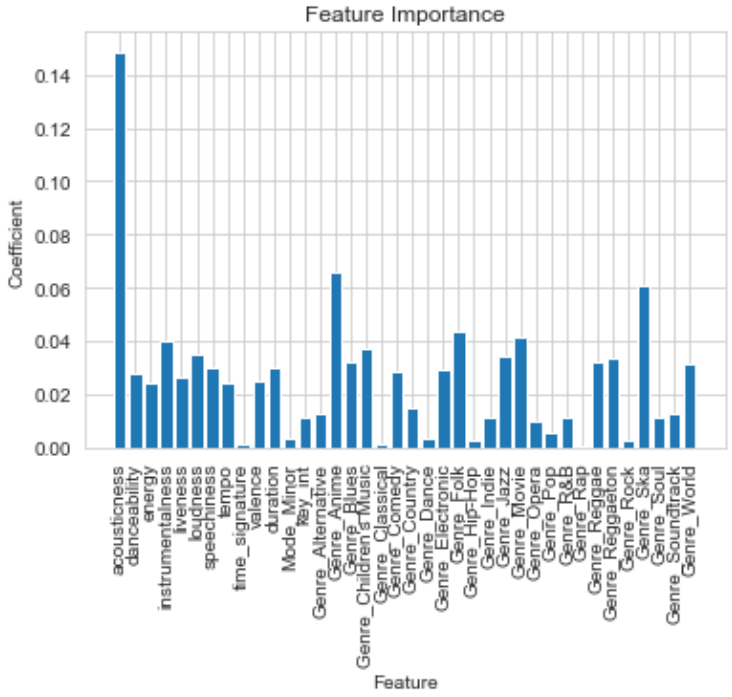

Figure 16. Importance of each feature for the bagging model

To find out if we could further improve our model, we calculated the importance of each feature in the bagging model and tried various combinations of feature selections. However, the model performed less accurately regardless of whichever features we chose to remove. Therefore, we were unable to improve the bagging model's accuracy any further.

5.3 KNeighbors

Next, we went with the theory that similar songs will most likely be of similar popularity and fitted a KNN model to our train dataset. To determine the best number of neighbors for our model, we iteratively ran the algorithm and compared the validation accuracies with different n_neighbors. It turned out that the model with n_neighbors = 51 made the most accurate predictions.

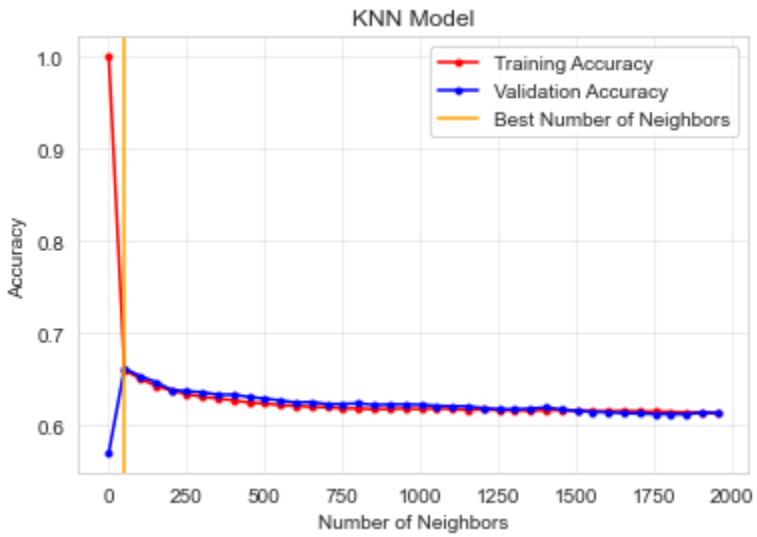

Figure 17. Train and validation accuracies at different n_neighbors

However, the KNN model took a long time to train due to the size of our dataset. The overall performance of the KNN model was also far from ideal. The accuracy scores of the model were too low (train accuracy = 0.6601 and validation accuracy = 0.6618) for us to continue working with it. Though we can't conclude that songs with similar features are not of similar popularity, we assumed that there was too much noise/randomness for our KNN model to perform its best.

5.4 Random forest

With a flexible model like decision trees where the model will memorize the training data and learn any noise in the data as well, overfitting can often take place. A random forest can reduce the high variance from a decision tree by combining many trees into one ensemble model. Even the trees in the random forest might all be mediocre, combining them into one ensemble is still better than a single tree.

Since our bagging model performed well, we also wanted to see how a Random Forest model compares in performance. We first used the RandomizedSearchCV method to fine-tune the hyperparameters of the random forest model. By iteratively fitting the train data, the cross-validation method determined that "max_depth" = 22, "n_estimators = 600", "criterion" = "log_loss", "min_samples_split" = 5, "min_samples_leaf" = 5, and "max_features" = "sqrt" make the best predictions.

Thus, we constructed a random forest model with the tuned parameters and fitted it to the train data. The values of the accuracy scores turned out to be 0.8598 (train) and 0.8272 (validation).

5.5 Neural network

Finally, we turned to Neural Network as our last model.

The idea behind neural networks is that there are a large number of nodes arranged into multiple layers. We have the input layer that takes in information regarding our data i.e. our features, and the output layer that tells us what our data is classified as i.e. our prediction. There are hidden layers in between the input and output layer where the model attempts to learn, recognize, process, and find patterns for our data. Figure 18 demonstrates this concept visually. Like our other models, neural networks rely on training data to learn and improve their accuracy over time.

The input layer of our neural network model contained 38 nodes, which corresponded to the 38 numerical and categorical variables in our train data set. There were also 3 hidden layers, containing 20, 100, and 20 nodes respectively. We also included a dropout layer of 0.2 where 20% of the nodes throughout all layers of our model are dropped to prevent it from overfitting. We set epoch = 20, where the entire training data is passed forward and backward through the data 20 times. The output layer is made up of only 2 nodes, representing the binary outcomes "mega-hit" and "mega-flop". The layout of the neural network is shown as below.

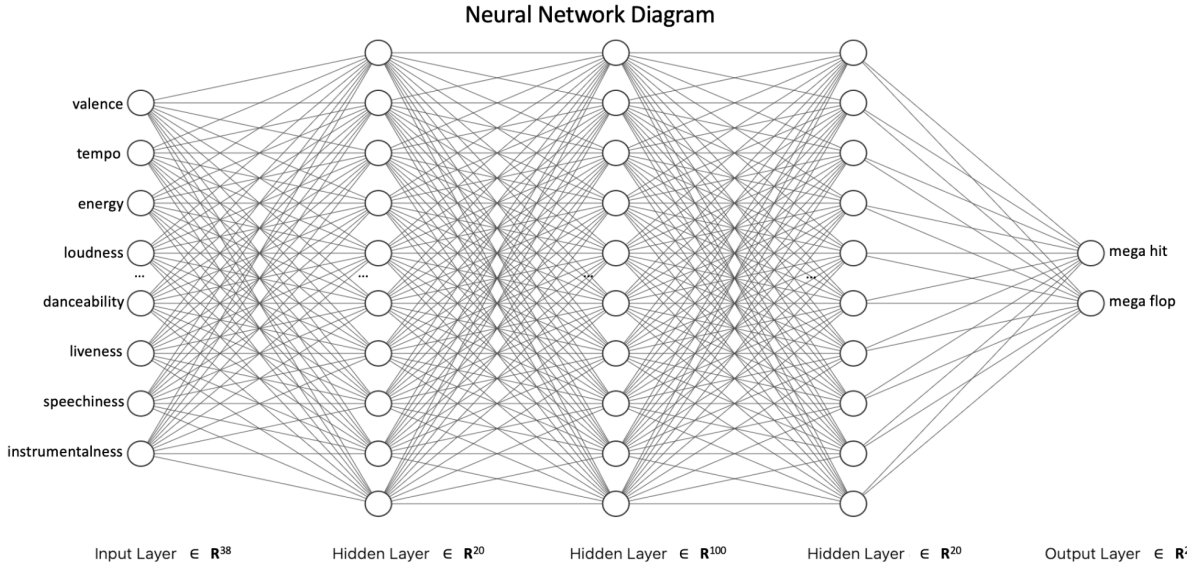

Figure 18. Neural network model diagram

The train and validation accuracies given by the neural network model were 0.8165 and 0.8208 respectively.

6. Multi-class Models

Our initial idea was to perform a binary classification where we predict whether a song is a "mega hit" or a "mega flop". However, we were also interested to see how well a multi-class classification could work on our dataset. We were excited to explore this new classification model as it allows us to predict a more specific range of popularity for each song. Hence, our target variable "hit_song" is now divided into 5 classes, namely "mega-flop", "flop", "average", "hit", and "mega-hit", with the cut-offs at every 20th percentile of the total number of songs (i.e. $0 \leq$ mega-flop $< 20\%$, $20\% \leq$ flop $< 40\%$, and so on). We have implemented this change by assigning new values to the column "hit_song" in our dataframe.

The percentage of each new class is visualized below.

	hit_song
mega-hit	21.295357
flop	20.546029
average	19.805301
mega-flop	19.208476
hit	19.144838

Figure 19. Percentage of each song category of the multiclass model

6.1 Multi-class logistic regression with hyperparameters tuning and features selection

With the same train and validation data, we fitted a multiclass logistic regression model to all the numerical and categorical variables. Our redefined variable "hit_song" is our predicted variable.

We first trained a logistic regression model with OVR as our training scheme, 'lbfgs' as our solver and 'l2' penalty just like we did before to obtain train accuracy = 0.5303 and test accuracy = 0.5340.

Using the same process we used for our binary classification, we used GridSearchCV to tune for the penalty term (L2 or no penalty) and the inverse regularization strength C of the logistic regression model. We continued to use the solver 'newton-cg'. We obtained the same C = 0.1 and penalty = 'none' that

gives the best results. With these parameters, we trained a logistic regression model that gave us train accuracy = 0.5327 and validation accuracy = 0.5438.

We also plotted a bar plot of feature importance for our multiclass logistic regression model.

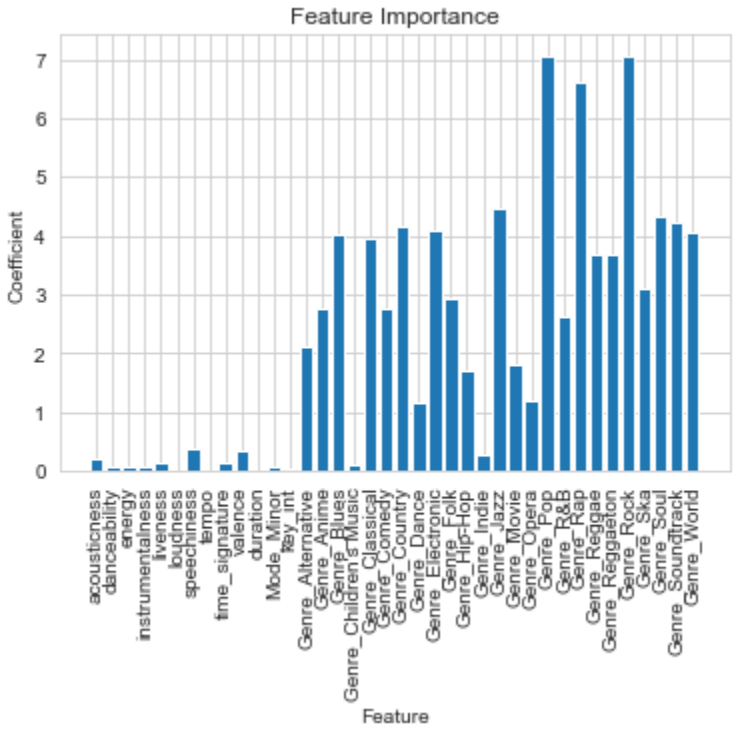

Figure 20. Importance of each feature for the multiclass logistic regression model

We noticed the same pattern in important features in our multiclass logistic regression model that we saw earlier with our binary logistic regression model. We decided to perform the same process of training our model only on the OHE encoded features and obtained train accuracy 0.5322 and a validation accuracy = 0.5389. Both the train and the validation accuracy dropped slightly, however, which is not what we expected. We figured that our model could actually be underfitting from observing the drop in train accuracy. We decided to move on to the other models since we saw promising results with the binary classification with those.

6.2 Decision tree with bagging

Same as we did with the binary classification, we first iteratively ran the decision tree algorithm to find the best depth, which turned out to be 19 for our multi-class case as shown in the figure below. The train and validation accuracies given by this model were 0.6772 and 0.4918.

Then we used bagging to improve the robustness of our model by reducing the variance.

Figure 21. Train and validation accuracies at different tree depths of the multi-class model

The bagging method improved the performance of the decision tree with train accuracy of 0.7286 and validation accuracy of 0.5360.

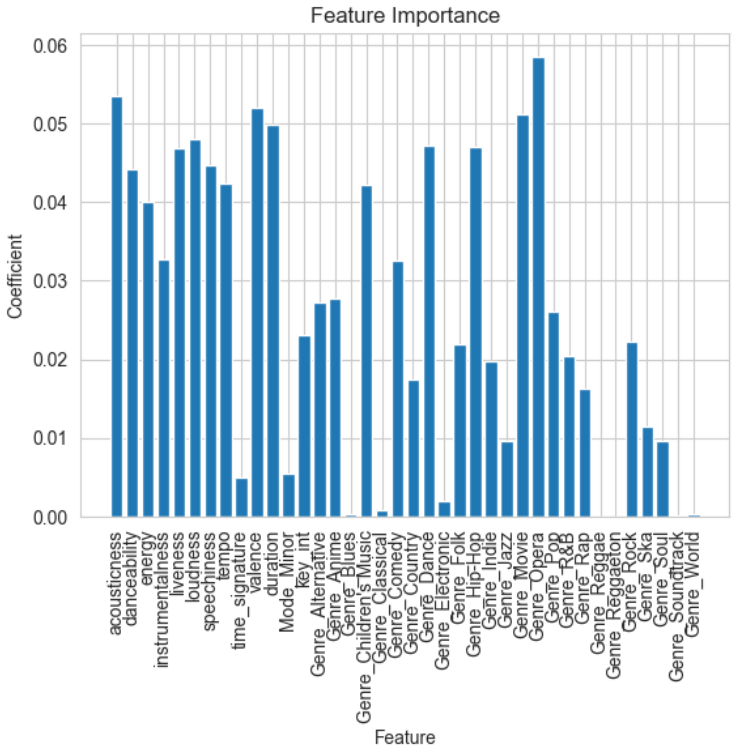

Figure 22. Importance of each feature for the multiclass bagging model

Same as the binary classification model, we tried to further improve the model by feature selections. However, we were unable to improve the Bagging model's performance any further.

6.3 KNeighbors

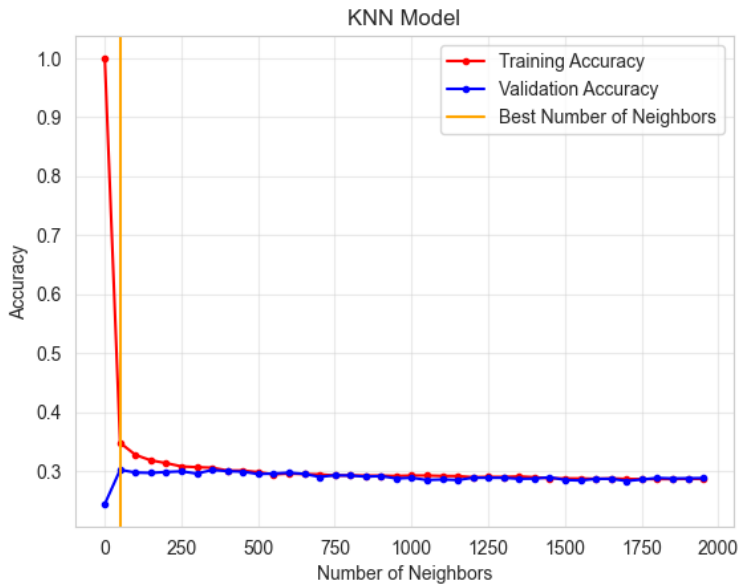

Figure 23. Train and validation accuracies at different n_neighbors

We then repeated the same process for the KNN model. The best number of neighbors for the multi-class model was the same as the binary case, which was n_neighbors = 51.

Similar to before, the KNN model performed poorly on our dataset as the train and validation accuracies turned out to be 0.3478 and 0.3020, respectively. Again, since it was time-consuming to train the KNN model and the model was not accurate, we did not see the point in keeping working with this model.

6.4 Random forest

We continued with the random forest model. Using GridSearchCV to find the best parameters, we found them to be “max_depth” = 18, “n_estimators” = 500, “criterion” = “log_loss”, “min_samples_split” = 7, “min_samples_leaf” = 5, and “max_features” = “log2”. The resulting random forest model produced train accuracy = 0.7315 and validation accuracy = 0.5360.

6.5 Neural network

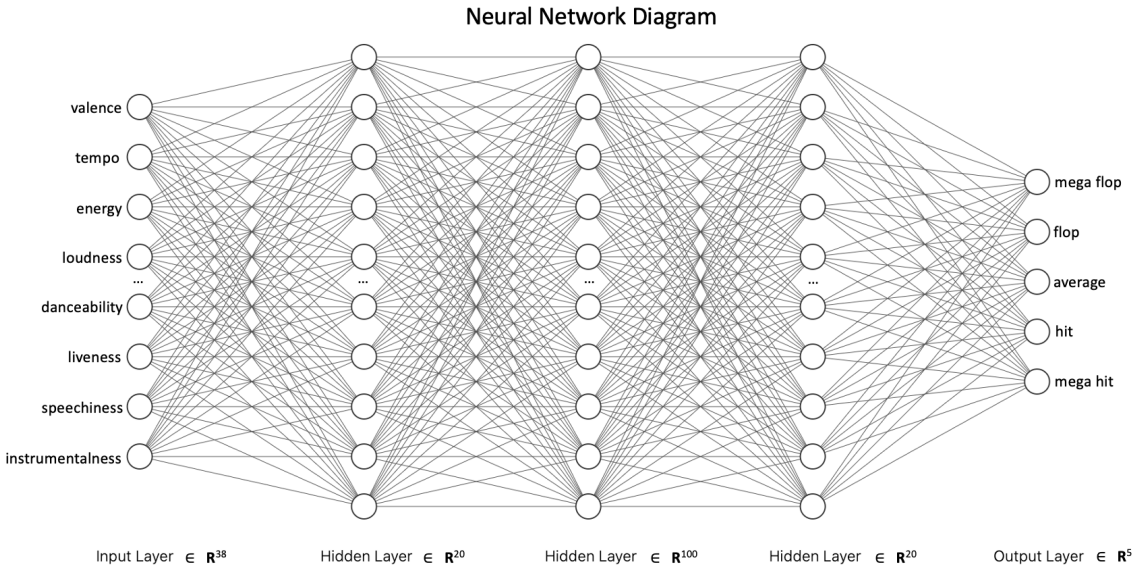

Figure 24. Neural network diagram of the multiclass model

At last, we built a neural network model for the multi-class dataset. In principle, the design of the model was the same as the one for the binary classification. The only difference was that the output layer contained 5 nodes instead of 2, corresponding to the 5 classes. The performance of the neural network was: train accuracy = 0.4972 and validation accuracy = 0.5070.

7. Conclusion

7.1 Overview of model accuracies

The train and validation accuracies of all binary classification models and multi-class models are summarized in the tables below. All tables are sorted based on validation accuracy in descending order.

	Model	Train Accuracy	Validation Accuracy
6	Random Forest	0.859819	0.827160
7	Neural Network	0.818332	0.823715
4	Bagging Best Depth	0.912791	0.821705
2	Logistic Regression Best Parameters + Feature ...	0.816825	0.820557
0	Logistic Regression	0.813738	0.819409
1	Logistic Regression Best Parameters	0.818332	0.819409
3	Decision Tree Best Depth	0.877907	0.807924
5	K-Nearest Neighbors Best K	0.660063	0.661786

Figure 25. Train and validation accuracies of various binary classification models

	Model	Train Accuracy	Validation Accuracy
1	Logistic Regression Best Parameters	0.532659	0.543784
2	Logistic Regression Best Parameters + Feature ...	0.532242	0.538903
4	Bagging Best Depth	0.728610	0.536032
6	Random Forest	0.731481	0.536032
0	Logistic Regression	0.530290	0.534022
7	Neural Network	0.497201	0.507034
3	Decision Tree Best Depth	0.677218	0.491817
5	K-Nearest Neighbors Best K	0.347761	0.302038

Figure 26. Train and validation accuracies of various multi-class models

7.2 Selecting a final model

At first glance, one might be tempted to conclude that the binary classification models outperformed the multi-class models as they generated far better accuracy scores. However, it is important to keep in mind that for a binary classification, the accuracy of making a complete random guess is 50% to begin with. On the other hand, there is only a 20% accuracy of randomly predicting an outcome out of 5 possibilities. Therefore, the models of both classifications improved the prediction accuracy by roughly 30%.

It is also worth noting that for both classifications, the Random Forest model and the logistic regression model with feature selection performed really well, though most of the top performing models are also extremely close in accuracy. Due to its consistency in accuracy with regards to both binary and multiclass classification, we concluded that the random forest classification is the best model for the data that we have. It is also one of the models that is relatively easier and quicker to tune and train with the help of RandomSearchCV.

7.3 Obtaining the test accuracies for all models

After selecting the random forest model as our preferred model, we scored all of our models with the test data that we set aside earlier.

	Model	Train Accuracy	Validation Accuracy	Test Accuracy
6	Random Forest	0.859819	0.827160	0.808955
4	Bagging Best Depth	0.912791	0.821705	0.813015
7	Neural Network	0.816466	0.820844	0.811507
2	Logistic Regression Best Parameters + Feature ...	0.816825	0.820557	0.812899
0	Logistic Regression	0.813738	0.819409	0.811971
1	Logistic Regression Best Parameters	0.818332	0.819409	0.813943
3	Decision Tree Best Depth	0.877907	0.807924	0.790164
5	K-Nearest Neighbors Best K	0.660063	0.661786	0.632989

Figure 27. Train and validation accuracies of various multiclass classification models

	Model	Train Accuracy	Validation Accuracy	Test Accuracy
1	Logistic Regression Best Parameters	0.532659	0.543784	0.532653
2	Logistic Regression Best Parameters + Feature ...	0.532242	0.538903	0.535321
4	Bagging Best Depth	0.728610	0.536032	0.528477
6	Random Forest	0.731481	0.536032	0.532769
0	Logistic Regression	0.530290	0.534022	0.532537
7	Neural Network	0.497201	0.507034	0.503770
3	Decision Tree Best Depth	0.677218	0.491817	0.491590
5	K-Nearest Neighbors Best K	0.347761	0.302038	0.295905

Figure 28. Train and validation accuracies of various binary classification models

Random forest did not perform the best out of all the models for our test data for both binary and multiclass classification. However, we stand by our decision since we would rather have a consistent model than a model that only performs well for certain types of classification. It is also worth noting that our random forest model did not require any feature selection to be one of our best performing models.

8. Future Work

8.1 Limitations and difficulties of our project

One of the main limitations of our project is that our dataset has one main categorical variable that needs to be one-hot-encoded in order to use the sklearn models that we wanted to try. The feature itself has 26 unique values. This resulted in a dataset with an already large number of observations to have even more features. We attempted to work with the full dataset, but we realized that with the time constraint, we would not be able to complete everything that we would like to try in time. We therefore made the decision to use a very small subset of our data to perform our training and testing on. One assuring

observation is that compared to the performance of the models that we were able to try with the full dataset, we saw very little difference with our model performance on the smaller dataset.

Another limitation is that due to the complexity of the features and the models that we have tried, our results are much harder to interpret. Especially for models such as decision trees and neural networks, it is hard to pinpoint the features that make a song a "mega-hit."

8.2 Strengths and future work

One of the strengths of our project is that we are able to get consistent results from most models with some form of feature selection or hyperparameter tuning. We are unable to make this work with our KNN model, but we definitely could have tried other forms of clustering given the time.

We would also have liked to attempt more feature engineering and feature extraction, especially with the track_id that is given in the original data. We felt that if we could extract the release date and add a new feature indicating the decade that the songs were released, we could have identified the popularity trends over time. We would have also liked to go further in depth as to how the introduction of more music streaming platforms (e.g. Spotify, YouTube Music, Apple Music etc.) correlates with any spike in popularities. We also would have loved to see how the boom in popularity of short-form media (i.e. TikTok, Instagram Reels, YouTube Shorts) could have resulted in a shift where songs with a shorter duration are more popular.

Other models that we would have liked to try are a support vector machine model and a k-means clustering model, which are both good for classification.

References

https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/

https://medium.com/nerd-for-tech/predicting-the-next-hit-song-f7c697f48647

https://www.explainthatstuff.com/introduction-to-neural-networks.html