CIS 520 Fall 2020, Project Report

Battle of the Billboards: Predicting Song Popularity on Spotify

Team Members:

- Catrina Hacker; Email: `cmhacker@pennmedicine.upenn.edu`
- Yuzhang Chen; Email: `yuzhangc@pennmedicine.upenn.edu`
- Ann-Katrin Reuel; Email: `akreuel@seas.upenn.edu`
- Jin Young Kim; Email: `jinykim@seas.upenn.edu`

Abstract

The recorded music industry makes billions of dollars globally each year. Streaming services such as Spotify, Apple Music and Pandora now make up a large part of that revenue. Their profit is determined by how many times a song is streamed. Thus, artists, recording labels and streaming services all have an interest in being able to predict which songs will be most popular to maximize their profits. We employed a number of supervised learning methods to predict the popularity of a song based on musical features scraped from Spotify. We focused on the ability of our classifiers to correctly classify the top 25% of songs in our data set. Starting with a single decision tree as a baseline with 47.72% test accuracy, our best model was Adaboost with 71.89% test accuracy in classifying the top quartile of songs. The predictions of these models could be used by members of the music industry to make investments in certain songs and artists over others.

1 Motivation

In 2019, the global recorded music industry's revenue exceeded 20 billion dollars. Streaming services accounted for over half of these profits, coming in at 11 billion dollars [1]. However, the income is not distributed equally between all artists. The majority of revenue in the recorded music industry comes from a small group of popular performers. In fact, the top 1% of artists are the source of over 60% of the revenue [2]. Thus, to maximize profit and maintain high user retention, it is important that recording studio, streaming services, and artists identify what songs are most likely to be successful.

Finding traits common to popular songs is a problem that can be solved by machine learning. Machine learning techniques can quickly process large numbers of musical features to determine which ones are most relevant to popularity and find relationships that might not be easily interpreted by human observers. In our final project, we will be using a dataset containing information about songs scraped from Spotify, the largest player in the streaming market[3]. We will use the scraped data to train several models aimed at predicting song popularity. Such algorithms would be of interest to music streaming services like Spotify as well as artists and recording labels hoping to predict which songs will become hits. These algorithms will allow streaming companies to proactively create contracts for new music and adjust payouts for existing artists when the contract is up for renewal. They will also allow recording labels to predict which songs will be hits so they put resources into artists and songs with the best chance of success. The ability to predict the popularity of a song is clearly of interest to many groups in the music world.

2 Related Work

The first related paper we found, *Hitpredict: Predicting hit songs using Spotify data*, was written by Elena Georgieva, Marcella Suta, and Nicholas Burton [4]. The authors built a model to forecast which songs will make it into the top 100 charts with an accuracy of roughly 75%. While they used logistic regression, GDA, SVM, decision trees and neural networks to approach the problem, only logistic regression and neural networks achieved this score on the test data. Despite some success in predicting song popularity, we identified two limitations of this approach: First, their models were only based on a training set of 4000 songs, which is rather limited to generalize findings on the song popularity given the diversity of song types and features. Secondly, their most predictive variable was "artist_score" which captures the familiarity with the artist. We found this an unsatisfying explanation as it relies on external factors and not just a song's inherent characteristics to make a prediction. While this might make sense for artists that have already released music, it would be unhelpful in determining the popularity of songs from new or unknown artists that recording labels might want to invest in.

In a second related paper, *Dance hit song prediction* [5], the authors attempt to predict whether a dance song makes the top 10 charts with an 85% accuracy on a polynomial SVM implementation. They also implemented a Naive Bayes method as well as logistic regression approach. The authors built their own database by incorporating songs from two different chart tables while limiting the selection of songs to the dance genre. Compared to our model, temporal features like timbre (capturing the tone color for each segment of a song) and beat diff (the time difference between subsequent beats) were available for the songs (but not in our dataset) that seemed to have some explanatory power.

Lastly, in the paper *Song hit prediction: predicting billboard hits using Spotify data* by Kai Middlebrook, and Kian Sheik [6], the authors achieved 88% accuracy using a random forest model to explain song popularity by audio features. While they used the same dataset as we will in our work, they didn't perform a feature selection and included explanatory variables *release_date* which captures the release date of a song. This feature is highly correlated with the popularity of a song: Given the time span that the Spotify dataset covers (1921 â 2020), and the fact that popularity scores were recorded in 2020, it can be assumed that more modern songs are automatically more popular (see section 5.1 for further explanations and feature selection process in our model). This implies a strong correlation with popularity that can't be traced back to inherent song characteristics. To eliminate the bias caused by this indicator, we excluded it from our analysis.

3 Data Set

We used a dataset called *Spotify Dataset 1921-2020, 160k+ Tracks*, which is available on Kaggle (`https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks`). This dataset is formmated as a csv file consisting of 169,909 songs released from 1921 to 2020, each of which has 19 features collected from Spotify using the Spotify Web API. The features are ID (id of track generated by Spotify), acousticness, danceability, energy, duration (ms), instrumentalness, valence, popularity, tempo, liveness, loudness, speechiness, year, mode, explicit, key, artists, release date, and name. The data are in multiple formats, which are numerical, categorical, or textual. Key is a categorical value taking on integer values between 0 and 11, mode and explicit are both dummy variables taking on values of 0 or 1, ID, name, and artist are all text data and the rest of the variables are continuous. Popularity, the value that we plan to predict, is a number between 0 and 100 describing the popularity of a song at the time of data collection based on number of streams and recency of those streams.

Although the dataset did not contain any missing values, we had to modify a few of the features. First, we eliminated id and name as these are unique to each song and will not provide any predictive power. We also eliminated the release date feature as many of the songs only had a year for release date and that is redundant with the year feature. We also eliminated the artist feature because most artists had only a single song, so this was unlikely to be a useful predictor. The key feature was the only categorical variable taking on integer values of 0-11. We changed this feature to be a one-hot encoding so there were twelve variables containing values of 0 or 1 depending on the value of key for that song. We formulated our goal as a classification task, so we grouped the popularity scores into four quartiles with an even number of songs in each and labeled them with 1-4 with 1 for the lowest 25% and 4 for the highest 25% of songs ranked by popularity.

Upon investigation of the correlation between each feature and popularity we found that year was highly correlated with popularity (r=0.88) and each year did not have a comparable distribution of popularity scores. This makes sense given popularity only measures the popularity of a song at the time of data collection. Initially, we trained our baseline model using all variables included. However, an analysis of the feature importances (details in methods) revealed that year was over-predictive of popularity. Based on this finding, we decided to exclude year to be able to focus on the inherent characteristics of a song. We also noted near zero importance for the key variables and mode so we removed those from the dataset to train the remainder of our models (see section 5.1 for a detailed explanation on feature selection). A correlation matrix for the features used for model training revealed that high energy and loudness were associated with more popular songs while those high in speechiness and acousticness were less popular. The full correlation matrix is given below.

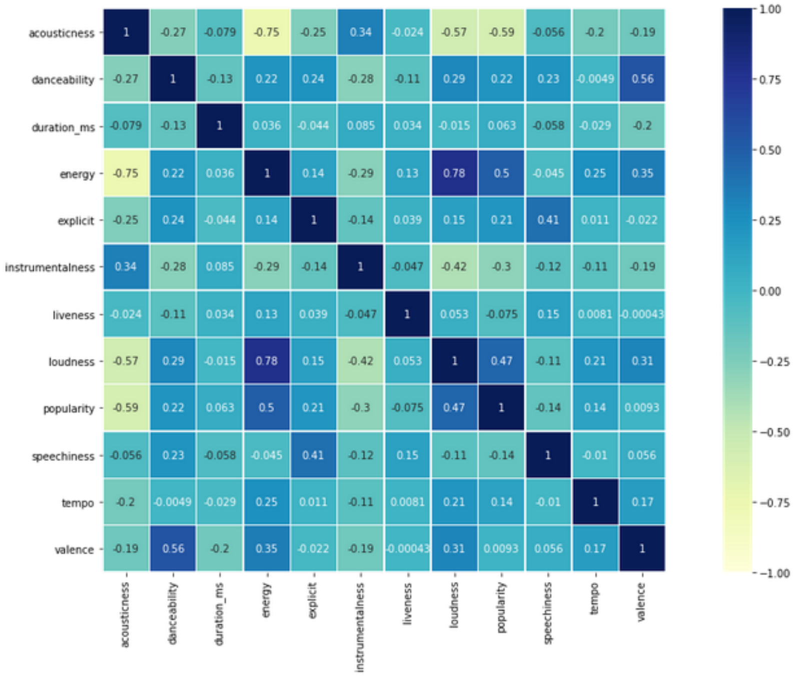

Figure 1: Correlation matrix showing the correlation between all of the features used for model training and testing and the target popularity score with each other.

To visualize our data we performed a principal components analysis and plotted the projection of each song onto the first two principal components. This visualization indicates that while the first and last quartile of popularity scores are relatively distinct from the other songs, songs that have middle levels of popularity are much less distinguishable along these first two PCs. Given that the goal of our project is to predict which songs will be most popular, we reasoned that these songs were sufficiently distinct in the PC space so as to be distinguishable by a classifier. This also foreshadows difficulties that we had with all of our models in reaching above chance classification of the middle quartiles.

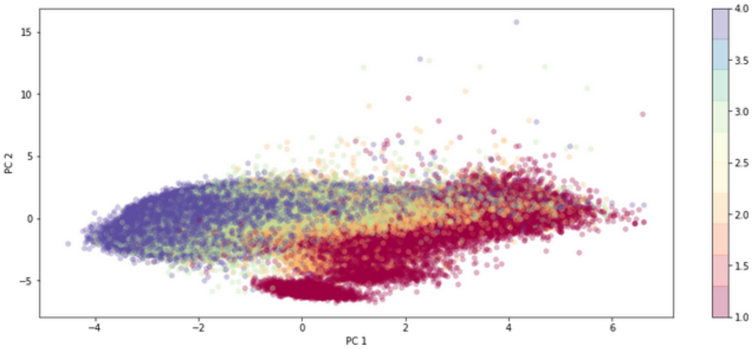

Figure 2: All 169,909 songs projected onto the first two principal components and labeled by color corresponding to their popularity quartile.

4 Problem Formulation

Both artists and recording labels have a vested interest in being able to predict which songs will be successful before they are released. For artists, such a prediction could save valuable time and money in writing, developing, and recording songs. For recording labels, these predictions could be useful in deciding whether to sign artists and how much to invest in a given artist or song. The goal of this project is to predict which songs are likely to be the most popular so that we can make recommendations as to which songs are worth investments of time and money given their higher chance of success.

We tackled this problem by using a handful of features that describe a song's acoustic properties (e.g., loudness, energy, speechiness, etc.) to predict their popularity. We reasoned that for a new artist or song the features of the song are the most likely available data points, as previous success for that artist might not be available. Further, predicting the popularity on only these features is more challenging than knowing a history of success for that artist, making this kind of model more desirable than one that relies on historical precedent. Given the vast majority of revenue in the music industry is made off a small percentage of the most popular songs, we focused our efforts on being able to predict these most popular songs.

After some initial difficulty in formulating this as a regression problem in which we attempted to predict the continuous value of each song's popularity between 0 and 100, we reformulated the task as a classification task. We sorted the data into four equal groups based on popularity and attempted to classify a song as belonging to one of these four quartiles. Based on our justification above, we optimized our models for classification of the fourth quartile (Q4) containing the top 25% of songs ranked by popularity although all of our models were trained to perform four-way classification. Given this goal, we employed models that are known to have high success in classification tasks such as decision trees and k-nearest neighbors. While the models we used for this project are focused on correlations and cannot make assertions about causation, this was a reasonable first step toward being able to predict whether an entirely novel song might be more or less popular based on these features. We make no assertions about what features should be built into a song to cause them to become popular as we did not test causality.

5 Methods

Our project focused on implementing several supervised learning methods to predict the popularity of songs based on their musical features. Given that the most popular songs are a small portion of all songs but are responsible for the vast majority of revenue in the music industry, we focused choosing classifiers that gave the best performance in classifying the top 25% of songs ranked by popularity in our dataset rather than overall classifier accuracy. **Throughout the report when we refer to classifier accuracy we are referring to accuracy in correctly classifying songs that were in the top 25% most popular songs in our dataset.** All methods were tested and tuned using the same training and testing splits of the data. An additional validation set was left out of hyperparameter tuning and selection and used to compute final test accuracy for each model. We reasoned that this would avoid accidentally overfitting to our test set as well as our training set as we selected hyperparameters optimized to that test set. Below are detailed descriptions of the justification and implementation of the methods employed in this report.

5.1 Baseline Model: Decision Tree

We chose a decision tree as our standard baseline model. Decision trees allow for non-linearity in the data, which we expect in the Spotify dataset since capturing the popularity of a song is an inherently complex task and none of our features had a particularly strong correlation with popularity. Furthermore, for a decision tree, no assumptions about the underlying distribution have to be made, which is in line with the fact that we don't have a prior with regards to the Spotify data. The decision tree was the base estimator for several other models that we employed (e.g., random forest, adaboost, gradient boosting) so it was an important baseline accuracy for comparison.

We used the *sklearn.tree.DecisionTreeClassifier* from the *scikit-learn* package with the respective default parameters. The most important features in this context are the maximum depth (max_depth) that we allow for the decision tree, i.e. the maximum number of child nodes that is allowed before the tree is cut off, as well as the minimum number of sample leaves (min_samples_leaf), which restricts the decision tree to having at least the specified number of data points in each leaf. The former is relevant because with an unlimited depth, as set in the default of the scikit decision tree classifier, we essentially allow nearly unlimited complexity, risking overfitting of our model. This is in line with the default value for the minimum number of sample leaves, which is set to 1 in the scikit implementation, because this effectively allows for unrestricted splitting as well.

Based on the default decision tree classifier, we computed the feature importances for the explanatory variables. Instead of calculating the standard impurity-based feature importance, we decided to use the permutation importance function from the *sklearn.inspection* package, because it tends to give more robust results than the impurity-based version.

In the permutation importance approach, after defining a baseline scoring, a single feature column from the test set is permuted. The importance is then defined as the difference between the baseline score and the one that was being calculated using the permuted column. Using this approach, we end up with the importance scores shown on the left of Figure 3:

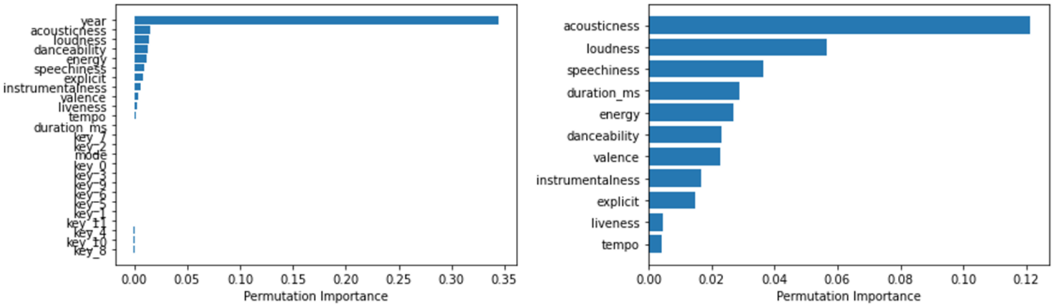

Figure 3: Left: Variable importance before feature selection, Right: Variable importance after feature selection

We decided to exclude all features with less than 1% explanatory power. This included all one-hot encoded key variables, as well as the mode variable. Furthermore, we found that the year variable was over-predictive. Given that this is an external feature that is not an inherent characteristic to a song, we decided to exclude it. The final variable importances can be found on the right of Figure 3.

All of the models described below used the updated training and test sets with these features eliminated.

5.2 Random Forest

Expecting our single decision tree to overfit, we decided to employ a random forest architecture to allow for a more nuanced model with managed complexity. A random forest in general can be seen as an extension of our baseline model, since multiple decision trees form the basis for this algorithm. We used the *sklearn.ensemble.RandomForestClassifier* from the *scikit-learn* package to implement the algorithm. The most important hyperparameters to adjust in this context are the number of trees in the forest (n_estimators), the number of features for splitting at the leaf nodes (max_features), the minimum number of sample leaves (min_samples_leaf) as well as the maximum depth of each decision tree (max_depth). As concluded in the previous paragraph, max_depth as well as min_samples_leaf will be adjusted to manage the model complexity while the number of trees will be tuned to increase the predictive power of the model.

Initially, we planned to use the *scikit* built-in *GridSearchCV* for hyperparameter tuning. However, we realized that we couldn't specify the accuracy that the different models were evaluated on. Given that our problem formulation entails a quartile prediction in which we want to optimize for the accuracy in Q4, we implemented a custom function that optimized the hyperparameters with respect to this accuracy score. Due to computational limitations and time constraints, we restricted the tested hyperparameter combinations to the following sets:

Parameter	Value
n_estimators	[10, 50, 100]
max_features	['auto', 'sqrt']
max_depth	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
min_samples_leaf	[1, 5, 10, 20, 50]

Table 1: Hyperparameters to be tested for the random forest model

5.3 Gradient Boosting

Staying in the realm of forests and trees, we suspected that a gradient boosting model would do reasonably well in terms of predicting Top-25% songs and better than a random forest. Inherently, gradient boosting is based on weak learner decision tree stumps. Initially, a stump is modelled to classify the data, before subsequent models focus on the prediction of cases that were inaccurately predicted by the first stump.

We make use of the in-built *sklearn.ensemble.GradientBoostingClassifier* from the *scikit-learn* package to build the model. There are two sets of hyperparameters that need to be adjusted in gradient boosting: tree-specific and boosting-specific parameters. Given that it's a greedy algorithm, we assumed that complexity needed to be handled to avoid overfitting, just like with the random forest and decision tree models. This is mostly easily accomplished by focusing on the tree-specific parameters, which are identical (in terms of functionality) to the random forest parameters explained above. However, compared to the previous random forest model, we restricted the maximum depth more for the gradient boosting: For gradient boosting, we want weak learners at each level â this is accomplished by restricting the maximum depth of each tree and hence the fitting potential of the model to the data.

Parameter	Value
max_features	['auto', 'sqrt']
max_depth	[2, 3, 5, 8]
min_samples_leaf	[1, 5, 10, 20]

Table 2: Tree-specific hyperparameters to be tested for the gradient boosting model

Additionally, the following boosting parameters were tested to increase the predictiveness of the model:

Parameter	Value
learning_rate	[0.01, 0.1, 0.5, 1]
n_estimators	[10, 30, 30]

Table 3: Boosting-specific hyperparameters to be tested for the gradient boosting model

Here, n_estimators is used to restrict the number of subsequently trained trees used. With regards to the learning rate, lower values are generally preferred since they account for a high robustness of the model, allowing for an accurate generalization. For values smaller than 1.0, the contribution of each tree added to the model is shrunk, too, resulting in a higher number of trees that needs to be added to make the model predictive. However, small learning rates are computationally expensive. Given the associated limitations we faced in this regard, we decided to focus on a small set of mid-sized learning rates. In addition to the described aspects, we decided to use the default scikit learn deviance loss function since its used for classification tasks with probabilistic outputs. Furthermore, the other available loss function in this context would have been an exponential one. However, this would have recovered the Adaboost algorithm which is discussed in the next chapter.

Just like with the random forest model, we make use of our custom hyperparameter search to allow for maximizing the accuracy of Q4 rather than the overall score that's used in *GridSearchCV*.

5.4 Adaboost

Adaboost is another ensemble machine learning algorithm. Initially, Adaboost begins with simple or weak classifiers and ensembles them into more complicated or stronger classifiers. At the end, the strongest classifier makes predictions. While Adaboost is capable of performing both classification and regression tasks, we used the Adaboost classifier to classify the songs with the top 25% of popularity scores. Adaboost classifier starts with a weak classifier to classify data into +1 or −1. The model usually uses a decision tree as a base estimator because the decision tree has advantages with regards to its performance and speed. Subsequentially, Adaboost gives more weights to misclassified data and uses another weak classifier to classify this newly weighted data. This process continues until the number of weak classifiers reaches the number of estimators. Then, when the number of weak classifiers reaches the specified number of estimators, all weak classifiers ensemble into a strong classifier, which will eventually make predictions. Here, we are using a the built-in implementation, *AdaboostClassifier*, which is available in the *scikit-learn* package in Python.

Since we are using the standard scikit-learn implementation, there are three major hyperparameters that need to be tuned to optimize the performance of the model: base estimator, number of estimators, and learning rate. The base estimator represents an algorithm that will be used in the training as a weak classifier. As mentioned above, a decision tree is most often used as a base estimator due to its fast performance. Thus, to tune the base estimator, we tried multiple decision tree algorithms with different depths. Number of estimators represents the number of weak classifiers or decision trees that will be produced. It is important to find an appropriate number of estimators in the algorithm because the Adaboost model could get too complicated due to many decision boundaries caused by a high number of estimators. Similarly, too few estimators might not reach a high enough level of accuracy. Learning rate indicates a rate that contributes to a weak classifiers' error minimization, which finds a decision boundary that produces a minimum error. In other words, it finds the minimum number of −1 given the dataset. This is an exponential loss function. Thus, there is a clear trade-off between the number of estimators and the learning rate, and it is important to control both hyperparameters appropriately. The tested hyperparameters are the following:

Parameter	Value
base_estimator	[DecisionTreeClassifier(max_depth=3), DecisionTreeClassifier(max_depth=4)]
n_estimators	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
learning_rate	[0.1, 0.01, 0.001]

Table 4: Table for testing hyperparameters

We manually searched for the best combination of hyperparameters that will result in the highest accuracy for the top 25% of popularity values (Q4). Additionally, in order to classify four classes appropriately, we are testing our base estimators, which are decision tree classifiers, with maximum depths of 3 and 4. Since we are doing a 4-way classification, the depth of the decision tree should be at least 3 so that there will be 4 terminal nodes at the end of the decision tree. We only sampled shallow tree depths to avoid the risk of overfitting with deeper trees. Once the hyperparameters were chosen, we computed each quartiles' accuracy.

5.5 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a computationally expensive but often very effective method for performing classification tasks that uses the observations closest to a sample to make a prediction. It is non-parametric and makes no assumptions about the distribution of the data. Because our data set is so large (n=169,909) we reasoned that any training set would represent enough of the space that we could obtain reasonable performance on a test set.

To implement this model, we used several functions from scikit-learn including *model_selection.KFold* for our k-fold cross-validation, *neighbors.KNeighborsClassifier* to implement the classifier and *metrics.confusion _matrix* to plot confusion matrices. Taking into consideration the long run times and computational cost of running k-nearest neighbors with this large of a data set, we performed 2-fold cross-validation to select the value of k that optimized the accuracy of classification of the top 25% most popular songs. Aside from using cross-validation to select the optimal value of the hyperparameter k we also tested three distance measures and two weighting schemes for the points. The distance measures we tested were the l_1, l_2, and l_{inf} norm and the weighting schemes we tested were uniform (no weights) and weighted by inverse distance. For each model we tested we computed classification accuracy for each quartile for a large range of k values (e.g., 1-1400). We noted that performance initially increased steeply and then tapered off before slowly falling again. We selected the value of k that corresponded to the top of this peak for classification of the top quartile of songs ranked by popularity for each of the models that we tested. This didn't always correspond to the absolute maximum test accuracy for the fourth quantile but because of the slow climb in accuracy after this point the computational cost of including thousands more neighbors outweighed the couple percent advantage in test accuracy. Selected k values ranged from 25 to 75. Because KNN is not scale invariant and the scale of the features in our set was not meaningful, we scaled the data before training or testing our models.

5.6 Support Vector Machines

Support Vector Machines (SVM) attempt to find a hyperplane in the feature space that separates the different classes of data with the largest margin. It is a general purpose classifier that tends not to overfit the data. Because the complexity of SVM increases linearly with the dimensions of the feature space and the number of output classes, SVM is more generally used for classification in lower dimensional spaces when there is a large number of training data. Our data has a relatively low number of output classes (4) and feature space (less than 20 features). Thus, we believe that SVM will be able to classify the large dataset in a reasonable amount of time with a reasonable degree of success. Further, the tendency of the SVM to not overfit, due to the hinge loss function, is attractive.

To implement this model, we used functions from the *svm* package in the *scikit-learn* package. We used two kernels: linear and radial basis function. The kernels define the decision boundaries. As such, we want to explore whether linear functions can better approximate our data than the complex radial basis function. To implement the linear kernel, we used *LinearSVC*. To implement the radial basis function kernel, We used *NuSVC* with the kernel parameter set to rbf. When using *NuSVC*, the parameter nu, which defines the upper bound on fraction of margin errors, was set to 0.5. We explored changing the parameter nu (0.25, 0.50, 0.75), but there did not appear to be significant changes to the accuracy.

Generally, the SVM iterates until convergence; however, the individual models did not converge in our implementation. As visualized in our principal component analysis, our dataset has a high degree of overlap in the primary principal components. This suggests that the data are not linearly separable. Thus, we set the maximum number of iterations to be 20 for *NuSVC* implementation and 100 for the *LinearSVC* implementation. These values were chosen because additional iterations (up to 100 with *NuSVC* and 2000 with *LinearSVC*) did not improve performance and more iterations take additional time. Data was scaled with the same reasoning as for KNN because SVM is not scale-invariant.

5.7 Multi-Layer Perceptron

A perceptron is a single layer neural network useful in binary classification. Combining multiple perceptrons into a multi-layer perceptron (MLP); however, converts the network into a universal function approximator when given enough neurons and layers. Because we can increase the dimensionality of the neural network in the intermediate layers, we can extract correlations in higher dimensions, at the risk of overfitting. Given the large amount of training samples, we believe we can increase the dimensionality somewhat without significantly overfitting to the training sample. Thus, we believe that the MLP will be useful in classifying songs according to popularity in a higher dimensionality environment.

To implement the model, we used the *MLPClassifier* function from the *scikit-learn.neural_network* package. MLPs use the cross entropy loss function. We started with a single perceptron (1) and then grew the network into one with up to 400 total perceptrons (100,200,100) spread across three layers. Intermediate steps include a single layer of 5 neurons, three layers of five neurons, three layers with 5, 10, 5, neurons in layers 1,2,3 respectively, and three layers with 25, 50, 25 neurons in layers 1,2,3 respectively. We also explored the effects of changing the default ReLU activation function to the Logistic activation function in two MLP models (5,10,5 and 25,50,25). To make the training time reasonable, we limited the maximum number of iterations to 200. Data was also scaled as it is a recommended step for neural networks. Normalizing data allows for faster convergence, speeding up the learning process.

6 Experiments and Results

In general, our models performed much better than chance but were below perfect classification. While we chose hyperparameters that would maximize classification accuracy for the top 25% of most popular songs, accuracy was always highest for the lowest quartile of songs and stayed very low for the middle two quartiles. Our models particularly struggled with classifying the third quartile, often misclassifying these songs as belonging to the second or the fourth quartile. Despite low accuracy scores, there was variability between our selected models and all of them were able to achieve classification accuracy well above the chance level of 25%.

6.1 Baseline Model: Decision Tree

Training the default, unrestricted baseline model resulted in a decision tree with depth 42 and a maximum number of features of 11, indicating a high model complexity. As suspected, this configuration overfits the data: With a training accuracy of 98.70 % compared to a test accuracy of 47.72 % for Q4, a significant difference between both scores can be observed, implying that the complexity of the model is too high for the given data. For a baseline model, the test accuracy is at a decent level though: it's reasonably above chance but leaves enough room for improvement.

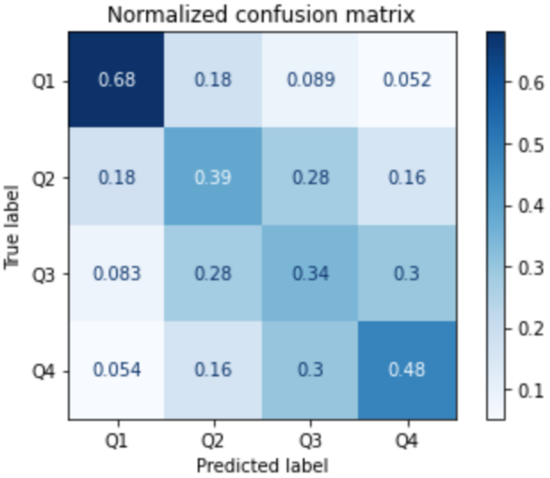

Figure 4: Confusion matrix of the decision tree model post feature selection

Even though this model suffered from overfitting, it fairly accurately predicted songs from the first quartile, indicating that songs of the lowest popularity are significantly distinct from songs in the second, third and fourth quartile in terms of characteristics. On the other hand, the Q4 predictions were only correct in 47% of the cases with high misclassification rates for predictions concerning Q3 and Q4, i.e. where Q3 got predicted as Q4 and vice versa.

6.2 Random Forest

We found that the test accuracy was highest (61.65%) for the following parameter combination:

Parameter	Value
n_estimators	100
max_features	sqrt
max_depth	10
min_samples_leaf	10

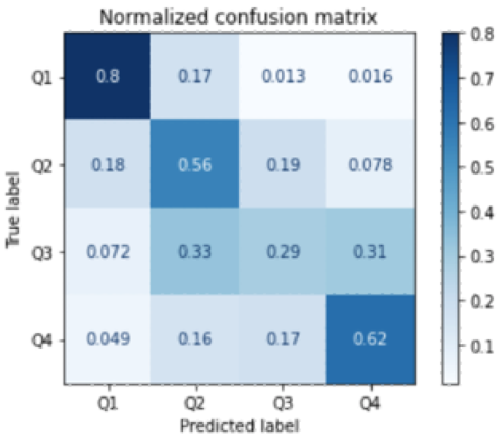

Figure 5: Left: Optimized hyperparameters for random forest, Right: Confusion matrix for optimized random forest model

This indicates that a relatively simple model is sufficient to achieve a reasonable accuracy, which was already implied by the significant difference between training and test accuracy we saw with the unrestricted decision tree. Restricting the complexity of the model limited the overfitting potential of the forest - this notion was confirmed by the small difference between training (64.03%) and test accuracy (61.65%). The corresponding confusion matrix implies that our algorithm predicts songs in the first and last quartile reasonably well, while quartiles 2 and 3 tend to be harder to correctly classify.

From the confusion matrix, we can see that the first and the fourth quartiles were predicted with a higher accuracy than Q2 and Q3. The lowest accuracy was achieved for Q3 where the songs were being mainly misclassified as Q4. We can further observe that between Q3 and Q4, the misclassified samples peaked, whereby the algorithm overpredicted Q4 when the true label was Q3.

6.3 Gradient Boosting

Running the described custom hyperparameter search for the gradient boosting algorithm with the combinations specified in the methods section, we found that a model with the following parameters performs best:

Parameter	Value
max_features	'sqrt'
max_depth	3
min_samples_leaf	5
n_estimators	30
learning_rate	0.01

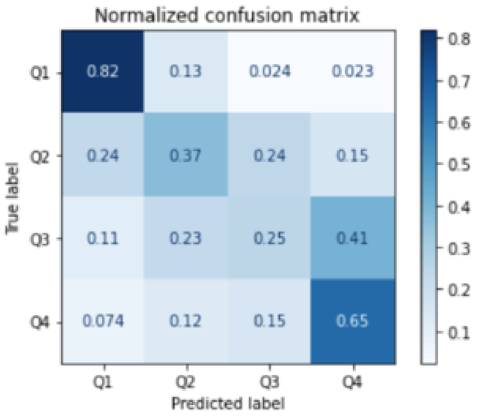

Figure 6: Left: Optimized hyperparameters for gradient boosting, Right: Confusion matrix for optimized gradient boosting model

This is in line with that we expected, given the initial results of our baseline decision tree which suffered from significant overfitting issues. When optimizing for the test accuracy, the model complexity should be controlled. This is done by selecting a higher minimum number of samples per leaf than the default (default = 0, hyperparameter search yielded 5). For the same reason, we expected the maximum depth of one stump to be limited. In the baseline model, a single decision tree ended up having a depth of 42, while the hyperparameter search for our gradient boosting model yielded a maximum depth of 3. The small learning rate was in line with our expectations, too: We want to only learn a limited amount from each new stump since by itself, each one is a weak classifier and not very predictive of the final outcome. This holds true for gradient boosting models in general. However, a decreased learning rate increases the run time of the algorithm, which also caused a trade-off for us. We couldn't try smaller rates because the hyperparameter search was already running multiple hours each time. The same holds true for testing a broader range of parameters â there might be an even better hyperparameter combination given broader testing which would have required more computational resources.

The test accuracy we achieved for this was 65.25% compared to a training accuracy of 65.12%. Given the proximity of both values, we can conclude that overfitting was not prevalent with the chosen parameter set.

Looking at the confusion matrix, the gradient boosting model mainly misclassified the intermediate quartiles. For Q1 and Q4, reasonably high accuracies were achieved. However, we can observe a high misclassification rate with regards to Q3 and Q4: The gradient boosting tends to have a significant amount of false positives in Q4 just like the random forest model.

6.4 Adaboost

With three significant hyperparameters mentioned above, we have a total of 60 combinations of Adaboost classifier models that we are testing to find the optimal combination of the hyperparameters. We present graphs of each quartiles' accuracy with multiple n_estimators and different learning rate and decision tree classifier. The four quartiles' accuracies with decision tree with a maximum depth of 3 are the following:

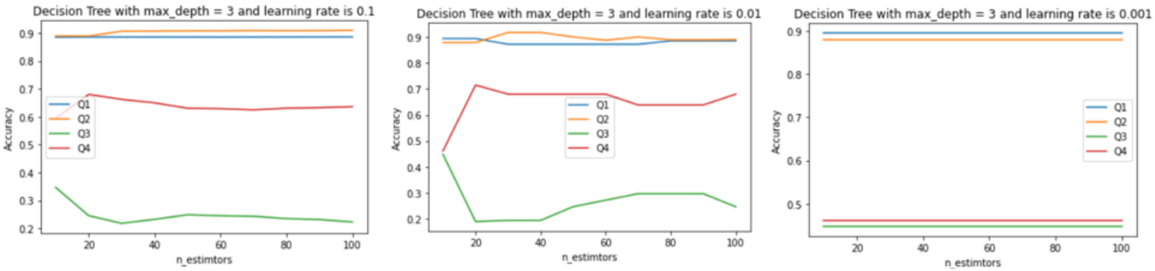

Figure 7: Four quartiles' accuracies calculated from DecitionTreeClassifier(max_depth=3) with different learning rates and n_estimators

The four quartiles' accuracies with decision tree with a maximum depth of 4 are the following:

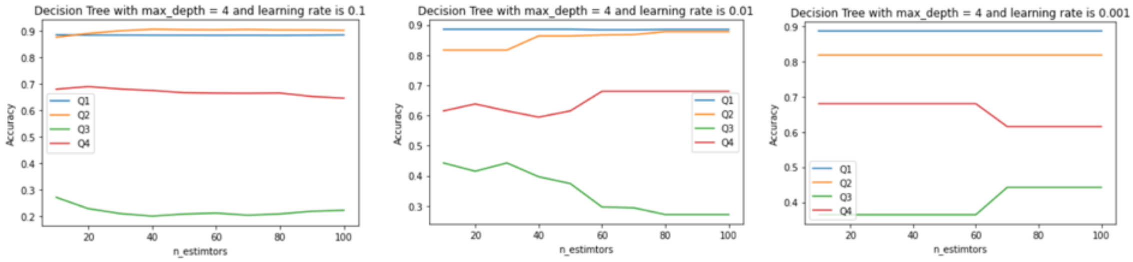

Figure 8: Four quartiles' accuracies calculated from DecitionTreeClassifier(max_depth=4) with different learning rates and n_estimators

From the outputs above, we know that the hyperparameters with the highest Q4 accuracy are:

base_estimator	n_estimators	learning_rate	Q4 Accuracy
DecisionTreeClassifier(max_depth=3)	20	0.01	71.50%

Table 5: Table for chosen hyperparameters for Adaboost Classifier

With the chosen hyperparameters shown above, we are able to plot a confusion matrix to validate this model.

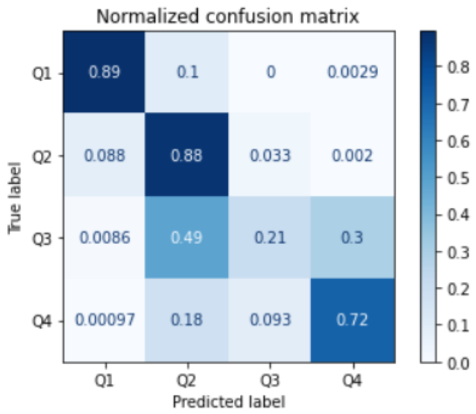

Figure 9: Confusion matrix with normalization for the adaboost classifier.

We can observe from the confusion matrices that our model is able to achieve a fairly high accuracy in classifying the first and second quartiles of data. The classifier also correctly classifies the fourth quartile of songs 72% of the time which is the highest level of accuracy achieved by any of our models in the report. The classifier struggled most in classifying songs from the third quartile, often misclassifying them as being from the second quartile. Encouragingly, despite low classification accuracy for this quartile, for all quartiles misclassifications were nearly always confusions with neighboring quartiles rather than larger mistakes (e.g., Q4 song misclassified as being in Q1).

6.5 K-Nearest Neighbors

For each of the six KNN models described in the methods section (l_1, l_2 or l_{inf} distance with uniform weighting or inverse distance weighting) we used the process explained in the methods section to choose a value of k that maximized classification accuracy for the top quartile of songs ranked by popularity.

Distance	Weighting	k	Training Accuracy	Testing Accuracy
l_2	uniform	25	62.13%	59.23%
l_2	inverse distance	35	98.48%	61.56%
l_1	uniform	60	63.40%	62.40%
l_1	inverse distance	75	98.71%	63.45%
l_{inf}	uniform	40	61.87%	59.99%
l_{inf}	inverse distance	50	98.71%	61.58%

Table 6: Optimal value of k, training, and testing accuracy (for classification of the top 25% of songs) for all of the KNN models that we tested.

All of the KNN models performed better than the baseline. The KNN models were most successful in classifying the least popular songs, but were also well above chance (25%) in classifying songs from the top quartile (reported above). The difference in performance between the worst of the six models and the best of the six models was only a 4.22% increase in classification test accuracy. The l_1 measure of distance consistently gave the best performance while the l_2 and l_{inf} norms led to comparable test accuracy. The models that weighted by inverse distance tended to overfit to the training data with performance on training data near 100% accuracy. Despite this overfitting, test performance of these classifiers tended to be slightly better than the models using uniform weighting and the same distance metric. Given the higher testing accuracy for these models despite apparent overfitting, we selected the KNN model using l_1 distance and weighting points by the inverse of their distance from the sample as our model for comparison. Below we show the confusion matrix for this model which is reported for model comparison in the discussion and conclusion section.

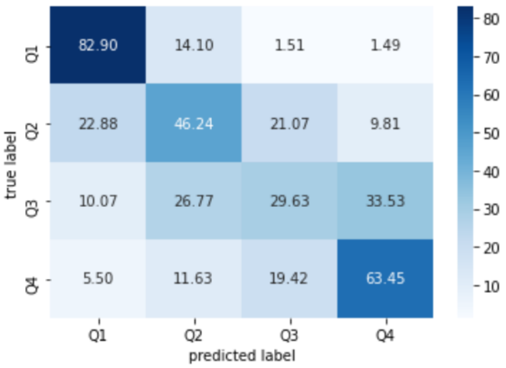

Figure 10: Confusion matrix for the best KNN model on the test data which used an l_1 norm to compute distance and weighted the distance of each observation from the sample by the inverse of that distance.

As can be observed in the confusion matrix above, the KNN model struggled to classify the intermediate quartiles. The third quartile had especially poor performance, with the classifer often misclassifying songs in the third quartile as belonging to the fourth quartile.

6.6 Support Vector Machine

We trained two support vector machines with different kernels. We also tried varying the number of iterations; however, accuracy did not change much once we went above 20 or so iterations. This is likely because the data cannot be cleanly split into distinct categories regardless of which divider we use. In addition, there are some benefits to having a lower number of iterations. This reduces the potential of the radial basis function kernel to overfit to the training set.

Model	Q4 Test Accuracy	Q3 Test Accuracy	Q2 Test Accuracy	Q1 Test Accuracy
Linear	66.07%	21.07%	33.35%	84.24%
Radial Basis Function	54.57%	30.24%	32.75%	71.77%

Table 7: Training and testing accuracy for Support Vector Machines.

The linear kernel underwent 100 iterations in *LinearSVC*. The radial basis function underwent 20 iterations using the *NuSVC* function. Because the linear separator performed much better than the radial basis function, we can assume that the data can be better separated by lines than an infinitely high dimensional function.

The evaluation metric we are using for the SVM is to find the highest Q4 accuracy. This is suitable because we are looking for only popular songs. Thus, the linear kernel is the best support vector machine out of all the combinations we tested. This stands even if we take into account the confusion matrix. Most of the songs that are incorrectly classified as Q4 come from the third quartile when using either kernel. In addition, true Q4 songs end up in Q3 more often than any other quartile in both models. The only advantage using a RBF kernel can give us over a linear kernel is that we have fewer Q1 and Q2 songs classified as Q4. However, that slight improvement does not offset a 11% decrease in accuracy for predicting true Q4 songs. Thus, the linear kernel is the better one out of the two support vector machine kernels.

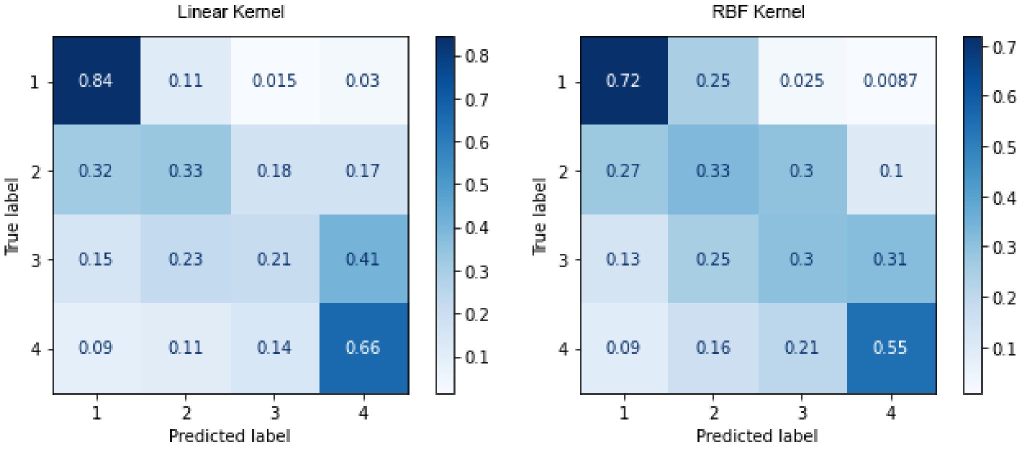

Figure 11: Confusion matrix for the support vector machines with linear kernel on left and radial basis function kernel on the right

6.7 Multi-Layer Perceptron

We trained eight MLP models. Initially, we performed parameter search on six MLP models containing only perceptrons with the default ReLU activation function. We started by evaluating the performance of a single perceptron, then increased to a layer of five perceptrons, three layers containing five perceptrons each, and then three layers with variable (5, 10, 5, or 25, 50, 25 or 100, 200, 100) neurons. Total number of iterations was hardcoded to a maximum of 200.

Model	Q4 Training Accuracy	Q4 Testing Accuracy
(1)	64.68%	65.40%
(5)	63.77%	63.89%
(5,5,5)	61.20%	61.48%
(5,10,5)	58.89%	58.74%
(25,50,25)	65.79%	64.36%
(100,200,100)	76.03%	53.40%

Table 8: Training and testing accuracy for ReLu Multi Layer Perceptron models.

We are focused on evaluating the performance of each model on the highest quartile (Q4) since identifying the most popular songs is the most important classification. If we focus on the training and testing accuracy for the highest quartile, we see from the table below that the highest performing model is the single perceptron followed by three layer network of 25,50,25 neurons.

On first look, it appears that the single layer perceptron performs the best. However, upon investigating the confusion matrix, we see that the one for a single perceptron is very different from the one for the three layer network. Both networks perform around the same for predicting which popular songs are in quartile 4 (most popular). However, the single perceptron network tend to overpredict quartile 2 and 3 songs as quartile 4. In contrast, the three layer network is better able to differentiate both quartile 2 and 3 songs from those of quartile 4. In addition, when the single perceptron misclassifies quartile 4 songs, they are roughly equally likely to be distributed into either quartile 2 or 3. For the multi-layer network, the misclassified songs are more likely than not to end up being classified as quartile 3. These slight differences indicate that the three layer network is better able to classify songs as most popular than the single perceptron network. I believe that improvement can be attributed to the higher dimensional processing in layer 2. I also believe it did not overfit as much as the (100,200,100) network because we had enough training samples to process the data at twice the dimension of the input features (50) but not at four times the input features (200).

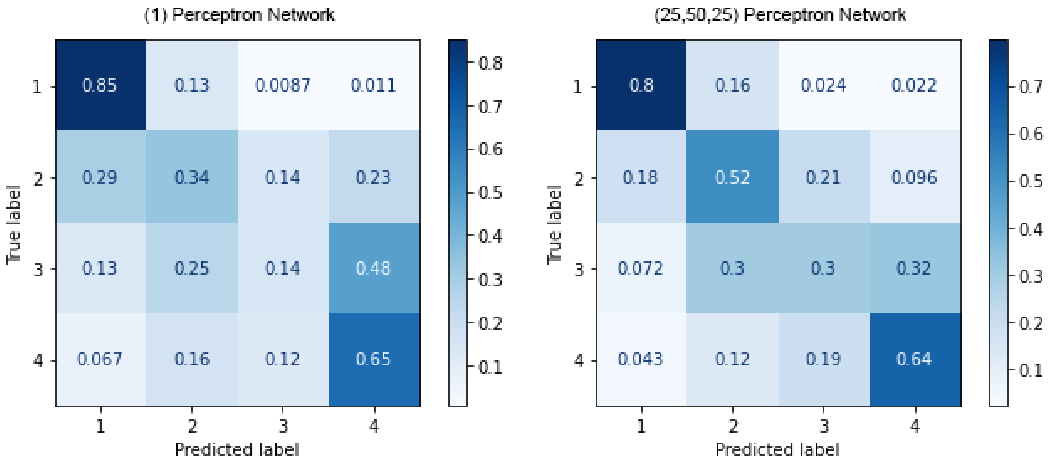

Figure 12: Confusion matrix for the single perceptron versus the three layer 25,50,50 multi layer perceptron shows that while accuracy in quartile 4 is similar, the single perceptron tends to significantly overpredict Q3 songs as Q4 compared to the multi layer network.

Now that we decided that increasing dimensionality can improve accuracy, we decided to test two of the models (5,10,5) and (25,50,25) with a logistic activation function instead of the ReLU activation function. Logistic activation functions penalize incorrect guesses more than the ReLu. While the accuracy in the higher quartile decreases, overall acurracy increases with the logistic activation function.

Model	Activation Function	Q4 Training Accuracy	Q4 Testing Accuracy	Overall Testing Accuracy
(5,10,5)	ReLU	58.89%	58.74%	55.31%
(5,10,5)	Logistic	59.13%	59.29%	55.70%
(25,50,25)	ReLU	65.79%	64.36%	56.51%
(25,50,25)	Logistic	57.72%	57.82%	57.30%

Table 9: Training and testing accuracy for ReLu and Logistic Multi Layer Perceptron models.

As we can see in the following confusion matrix, the accuracy for quartile two and three improved significantly due to the logisitic activation function. However, overall accuracy for quartile 4 and 1 decreased when compared to the network with ReLU activation function. The other trends mentioned in the earlier discussion with the ReLU models remain unchanged.

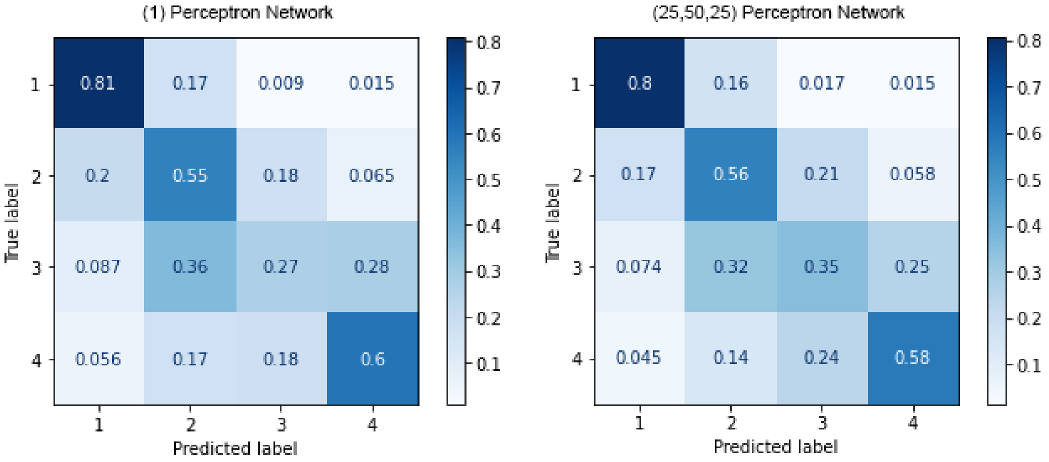

Figure 13: Confusion matrix for the single perceptron versus the three layer 25,50,50 multi layer perceptron. Both of them have logisitic avtivation function as opposed to the ReLu in the earlier figure.

As summary, if we were looking for the best network for classification for each class, we would choose the three layer (25,50,25) MLP with the logisitic activation function. However, if we are looking for the best network for classifying the highest quartile, we would choose the three layer (25,50,25) MLP with the ReLU activation function. This is because projecting the dataset to a higher dimension allows for both better separation of Q4 songs from Q3 songs and for more incorrectly classified Q4 songs to end up in Q3 rather than lower quartiles.

7 Conclusion and Discussion

We used seven supervised learning methods to predict a song's popularity from its musical features. Specifically, we assessed the ability of four-way classifiers to successfully distinguish the top 25% of songs from other quartiles. All six methods beat the baseline single decision tree's accuracy of 47.72% with chance performance at 25%. Our best method achieved 71.89% test accuracy indicating space for further exploration and improvements in the future.

Model	Training Accuracy	Testing Accuracy
Decision tree	98.70%	47.72%
Random forest	64.03%	61.65%
Gradient boosting	65.25%	65.12%
Adaboost	71.15%	71.89%
K-Nearest Neighbors	98.71%	63.45%
SVM (linear)	66.06%	66.07%
SVM (RBF)	53.53%	54.57%
MLP	65.79%	64.36%

Table 10: Training and testing accuracy for all of the models on the top 25% of most popular songs.

7.1 Model Comparison & General Insights

Our highest performing model was Adaboost with 71.89% classification accuracy for the top 25% of songs. This aligned well with our initial intuitions that ensemble methods would outperform most singular models. Both Adaboost and gradient boosting outperformed the single decision tree and even the random forests.

Surprisingly, the linear kernel SVM performed better than the gradient boosting method. This suggests that the linear solution to which the SVM converges describes the relationship between these features and popularity better than the more complex interactions that can be modeled by decision trees.

In general all of our models were most successful in classifying the top or bottom 25% of songs while they struggled to classify the middle two quartiles. We provided confusion matrices for all of our methods that show that in many cases the low classification accuracy for songs in the third quartile is because they are mistakenly classified as being in the top quartile. This could be due to how we divided the data. Few songs are very popular, so some of the songs that belong to the fourth quartile might not be well representative of the most popular songs. Instead, they might better match the features of songs that are currently classified in the third quartile. This would explain why differentiating between songs in the third versus fourth quartile would be especially difficult.

Across all our considered models, we found that the most unpopular songs (Q1) were more easily predicted, as indicated by the high accuracy scores in the upper left corners of the confusion matrices. One possible explanation for this is that they might share specific inherent features that make them less popular. On the other extreme, this seems to hold true for songs in the top 25%, too, albeit not to the extend of song feature similarity as in the first quartile since the confusion matrices showed a higher classification accuracy for Q1 than Q4. While the features for this group of songs don't seem to be as distinct as for the most unpopular ones, the higher accuracy scores compared to Q2 and Q3 indicate that our considered features must have some predictive power with regards to a song's popularity.

The difference between the Q1 and Q4 accuracy could furthermore be explained by the fact that in Q4, not only characteristics inherent to a song are relevant to popularity but also other external factors like artist familiarity which we didn't account for. This could explain the high classification error for songs being classified in Q4 when their true label is Q3: at this level of popularity, song features alone are not predictive enough of the overall popularity anymore because external factors like familiarity with a band start to play a more significant role. Not including these features was in line with the primary goal of our project: we wanted to develop a model that would allow producers and artists to get an idea whether the song in question can make the charts based on its inherent musical features, even without additional "boosters" like marketing efforts by the label or for an entirely new song or artist.

Lastly, we want to comment on the fact that the accuracy scores for Q1 and Q4 were still significantly above the ones for Q3 and Q4 in our models. This indicates that while songs in the first and last quartile tend to share common characteristics in their respective groups, songs of intermediate popularity are more variable. In general, we found that there was a significant amount of variance in popularity that our models couldn't explain which implies that there have to be other features of songs or factors that influence song popularity apart from the ones considered in our models.

7.2 Lessons learned

From this final project, we learned that it is important to think about the appropriate model and the range of hyperparameters for each model carefully in advance. Despite many models being classified as "classifiers" (e.g. K nearest neighbors and SVM), not all models perform equally well in every case. As an example, the worst SVM scores an accuracy of 53.53% while the K-Nearest Neighbors performs at 63.45%. In addition, we learned that it's important to critically think about appropriate ranges of hyperparameters since it is time (and resource) consuming to conduct an exhaustive search of all possible combination of parameters.

In addition, it is necessary to find a cohesive dataset for the task at hand. Features that are included in our dataset are more relevant to musical genre classification; however, we were still able to find interesting relationships because we had a large training set. While a small number of features gave us a manageable starting point and would allow for a more explainable model if it would be used in industry by producers, including more features could have also increased predictive power in our models. This to show how important it is to have a large dataset when making a unique interpretation of existing datasets.

7.3 Future directions

One of the limitations in our project was the lack of information given the limited number of features available. From the results from eight different models shown above, we realized that the data in the first quartile tended to be classified easily, whereas the data in the second, third, and fourth quartiles showed poorer performance with regards to the classification. Although there were 11 features used in this classification project, the number of features might not be sufficient to correctly classify the data in all four quartiles. By adding more predictive features, either external ones like social media presence as an approximation of artist familiarity or additional musical characteristics, we could expect better results in classifying popular songs.

In general the poor performance on the fourth quartile could be attributed to misclassifying these songs as belonging to the third quartile. Future models could optimize for sensitivity instead of accuracy so that models are penalized for both misses and false alarms in the fourth quartile.

We currently have coarse divisions in song popularity, but ultimately we would like to be able to predict which songs will be in the very small percentage of most popular songs. Future work could reduce target percentages. Instead of classifying top 25% songs, classifying the top 10% of songs can narrow and make more specific target populations which might have more practical applications.

Finally, we can implement more advanced models, such as neural networks. For instance, if we can get data related to song lyrics, then we could use some natural language processing techniques to classify popular songs based on content, or if we could make the music data into graphical data or explore the impact of album art or music videos on popularity, then we could also use CNN or other computer vision techniques to classify popular songs as well.

Acknowledgments

We would like to thank Dr. Ungar for his help as we developed the project.

References

[1] IFPI. *Global Music Report: The Industry in 2019.*. London. 2020.

[2] Lunny, Oisin. *Record Breaking Revenues In The Music Business, But Are Musicians Getting A Raw Deal?*. Forbes, 15 May 2019.

[3] Trainer, David. *It Sounds Like Spotify Is In Trouble*. Forbes, 13 Oct 2020.

[4] Elena Georgieva, Marcella Suta, and Nicholas Burton. *Hitpredict: Predicting hit songs using spotify data.* `http://cs229.stanford.edu/proj2018/report/16.pdf` 2018.

[5] Dorien Herremans, David Martens, and Kenneth Soerensen. *Dance hit song prediction.* Journal of New Music Research, 43(3): 291-302, 2014.

[6] Kai Middlebrook, Kian Sheik. *Song hit prediction: predicting billboard hits using Spotify data.* `arxiv.org/abs/1908.08609`.

[7] Rutger Nijkamp. *Prediction of product success: explaining song popularity by audio features from Spotify data.*
`https://essay.utwente.nl/75422/1/NIJKAMP_BA_IBA.pdf`.