DS 4400: Machine Learning and Data Mining I

Spring 2022

Final Project Report

A Machine-Learning Approach To Song Recommendation

TA: Nate Hofmann

Team Members: Payton McAlice & Sonia Popovic

Link to code:
https://drive.google.com/file/d/1ffHsLHF9Tvlf75YYWJerHF_SUv5sW1FO/view?usp=sharing
Link to video recording:
https://drive.google.com/file/d/1MFhBO8RkcQezApag_2fONM2Zsa5m5e2J/view?usp=sharing
Link to presentation:
https://docs.google.com/presentation/d/1YPhL0uEG0ZDd24Ui3yQzkCDergZ0agdnsEQ6dElsP8g/edit?usp=sharing

I: Problem Description

An integral part of every music streaming service is its system of recommending new songs to its user base. Most modern platforms such as Spotify rely on the implementation of machine learning models to select new songs that a user will likely enjoy based on their existing listening habits. While popularity, artist, and genre certainly play a role in what songs are recommended, audio features that describe the sonic attributes of a song are also considered. To help gain an understanding of what features are important in the song-recommendation process, our machine learning project looks to implement a song-recommendation system using actual song and user data from Spotify. We would like to gain insight into the music recommendation process and test the effectiveness of different machine learning models and algorithms in regards to finding songs that users will enjoy.

With our approach to this project, the problem we are presenting is a classification problem. We have a binary classification of songs a user would enjoy and songs a user would not enjoy based on their existing listening habits. We will create these classifications using Spotify data of a user's listening history, as well as an online dataset of several thousand songs and their features. Songs pulled from specific users will be those that said user has expressed interest in via saving or favoriting, and this will give us the training data for songs that the user is known to enjoy.

The importance of this project is in improving the listening experience for music fans as well as providing insight into what possibly helps in shaping the music taste of an individual. With the abundance of music readily available to the public, exploration of new music can seem overwhelming. Having a model that is effective at finding music that a user will enjoy independent of said music's popularity can help personalize the user experience to a new level. Additionally, having a model that we can look at under the hood gives us insight into what defines our own personal music tastes, which is interesting if nothing else and could help us better understand what music we find ourselves enjoying on an individual level.

II: References

The following are some resources of past research and projects of a similar nature:

- https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022
    - A resource describing some techniques currently used in Spotify's song-recommendation system, including content-based and collaborative filtering, techniques we struggled to implement in this project. Also describes some of the data used and NLP used on song lyrics.
- https://www.section.io/engineering-education/building-spotify-recommendation-engine/
    - A project that creates a Python program that uses content-based and collaborative filtering to recommend songs to a user. Uses Spotipy to have easier and consistent access to Spotify compared to simply making Spotify API calls.
- https://thecleverprogrammer.com/2021/03/03/spotify-recommendation-system-with-machine-learning/
    - A project that creates a Python program that uses a distance metric to find songs similar to those liked by the user. Does not use any of the models we learned in class.
- https://jiading-zhu.medium.com/1-intro-1b6e3a4b2fb3
    - A project that examines Spotify song data in a song-recommendation context and implements regression and decision trees to assess song-recommendation as a classification problem. Also used accuracy when choosing and tailoring a model.
- https://blog.mlreview.com/spotify-analyzing-and-predicting-songs-58827a0fa42b
    - Another example of a song-recommendation project that implements models including regression, random forests, and k-nearest neighbors. Similarly used ROC curves and correlation matrices to visualize data and model metrics.
- https://github.com/lognorman20/spotify_recommender/blob/main/model_creation.ipynb
    - Spotify machine-learning that uses SMOTE and a sample of songs from Spotify's collection of default playlists as well as user-liked tracks.

III: Dataset & Exploratory Data Analysis

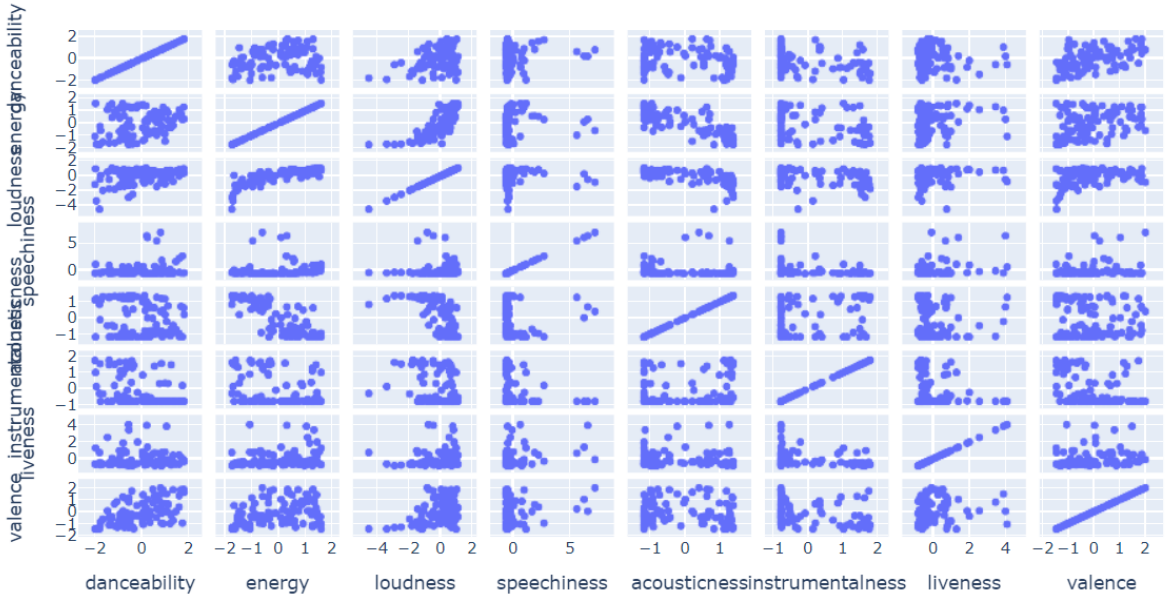
Our dataset comes from two different sources. The first source is a 20.4 MB public dataset from Kaggle with more than 130,000 songs and their audio features. All of the songs and their features were scraped directly using Spotify's API. There exists data about the song itself (ID, title, artist name, etc.) as well as values corresponding to the attributes of the song (energy, acousticness, etc.). These songs were classified in our models as class 0; which indicates songs that we cannot confidently say a given user likes. This dataset was combined with user-specific data scraped directly through the API and a generated authentication token. These songs were those classified as "top tracks" or "liked tracks." The songs were all scraped with the same features as were available in the Kaggle dataset. They were classified in our models as class 1; which indicates a song that we know a user enjoys based on their own listening history and indications.

Below is a list of the audio features that were collected and used in our machine-learning models as well as their descriptions as provided by Spotify:

- *Danceability:* "How suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity"
- *Energy:* "A perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy"
- *Key:* "The key the track is in. Integers map to pitches using standard Pitch Class notation"
- *Loudness:* "The overall loudness of a track in decibels (dB)"
- *Mode:* "The modality (major or minor) of a track, the type of scale from which its melodic content is derived"
- *Speechiness:* "Detects the presence of spoken words in a track"
- *Acousticness:* "A confidence measure from 0.0 to 1.0 of whether the track is acoustic"
- *Instrumentalness:* "Predicts whether a track contains no vocals"
- *Liveness:* "Detects the presence of an audience in the recording"
- *Valence:* "A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track"
- *Tempo:* "The overall estimated tempo of a track in beats per minute (BPM)"
- *Duration:* "The duration of the track in milliseconds"
- *Time signature:* "An estimated time signature"

When working with song data, variable independence and correlation was of particular interest to us. Since songs often have patterns based on what may be sonically pleasing, we figured that there was potential for strong correlations between variables. To examine this, we created a correlation matrix for all of the song features for every song both in our Kaggle dataset and our user-specific dataset. For the latter, it is worth noting that correlations may vary depending on which songs comprise the given user's taste in music. However, in the Kaggle dataset, we were able to identify areas of strong correlation.

	danceability	energy	key	loudness	mode	speechines	acousticne	instrumenta	liveness	valence	tempo	duration_m	time_signature
danceability	1.00	0.28	0.02	0.38	-0.02	0.21	-0.29	-0.29	-0.04	0.56	0.06	-0.14	0.20
energy	0.28	1.00	0.03	**0.82**	-0.03	0.13	**-0.80**	-0.25	0.21	0.40	0.27	-0.04	0.19
key	0.02	0.03	1.00	0.02	-0.15	0.02	-0.02	-0.02	0.00	0.02	0.01	-0.01	0.01
loudness	0.38	**0.82**	0.02	1.00	0.00	0.06	-0.67	-0.40	0.14	0.38	0.25	-0.05	0.18
mode	-0.02	-0.03	-0.15	0.00	1.00	-0.05	0.05	-0.05	0.01	0.04	0.01	-0.03	0.00
speechines	0.21	0.13	0.02	0.06	-0.05	1.00	-0.08	-0.17	0.19	0.10	0.01	-0.08	0.02
acousticne	-0.29	**-0.80**	-0.02	-0.67	0.05	-0.08	1.00	0.25	-0.12	-0.27	-0.23	0.00	-0.18
instrumenta	-0.29	-0.25	-0.02	-0.40	-0.05	-0.17	0.25	1.00	-0.12	-0.29	-0.07	0.12	-0.07
liveness	-0.04	0.21	0.00	0.14	0.01	0.19	-0.12	-0.12	1.00	0.07	0.03	0.00	0.02
valence	0.56	0.40	0.02	0.38	0.04	0.10	-0.27	-0.29	0.07	1.00	0.18	-0.19	0.15
tempo	0.06	0.27	0.01	0.25	0.01	0.01	-0.23	-0.07	0.03	0.18	1.00	-0.02	0.07
duration_m	-0.14	-0.04	-0.01	-0.05	-0.03	-0.08	0.00	0.12	0.00	-0.19	-0.02	1.00	0.03
time_signat	0.20	0.19	0.01	0.18	0.00	0.02	-0.18	-0.07	0.02	0.15	0.07	0.03	1.00



Per the correlation coefficients, we found a strong positive correlation between loudness and energy and a strong negative correlation between energy and acousticness. Intuitively, these make sense. We ultimately decided to use all of our available features in model training and

implementation, as these general correlation trends may not hold necessarily true when also factoring in an individual's music preference. These correlation coefficients were calculated after the standardization of each variable.

As a result of the limitations of Spotify's API and the data available through it, we were only able to get up to 100 songs that we were confident in a given user liking. This created a massive imbalance between our 0 class and 1 class. In our approach to training and testing, this imbalance is addressed.

IV: Approach & Methodology

We began by retrieving sample data using one of our own Spotify API codes. From there, we were able to match and clean up the dataset from Kaggle to match Spotify's feature list. We decided to eliminate a few of the song features that we had deemed unnecessary for recommending a song. These included 'track name', 'artist name', 'release date', 'song id', etc.

Now that we had our data, we needed to create a target class that our model would aim to predict correctly. So, to both datasets, we added a column 'liked' where 0 indicated that we did not know whether the user liked a song and 1 indicated that they did. Since we knew that the songs we had gathered from Spotify were listened to and enjoyed by the user, we gave all of these a value of 1 to begin with. Similarly, with the Kaggle dataset, those received a value of 0.

Since Spotify API can only return 50 results and we were only using two API calls, our training and testing datasets would be incredibly unbalanced with 100 for training and 100,000 for testing. To fix this problem, we used SMOTE to balance class distribution by adding more songs into the 1 class. Now, our model would have much more data to learn from. We randomly split our data into training and testing samples at an 80-20 ratio.

From here, we tested and compared a few different models: logistic regression, Naive Bayes, decision tree, random forest, AdaBoost, and neural network. We noticed that for all of these models, the accuracy was high in the 90% range but the precision was extremely low, under 1%. We understood however, that since we had such a large dataset to predict and such a small training set, most models would over predict a user liking a song. For that reason, we decided to focus on the accuracy score and AUC score when choosing which model worked best. It seemed that AdaBoost had the highest scores for both of these categories.

We then created a function playlist_generator that would take in a user's known liked songs as well as a dataset of unknown songs to choose from. This function would return 10 songs that have been chosen by the model. We decided to use predict_proba in conjunction with AdaBoost so that we could select the 10 most likely candidates for being enjoyed by the user.

Here we decided that to truly test our model, we would need to get real feedback from real users on this predicted playlist as we would otherwise have no idea how well our model

performed. Thus, we reached out to a few of our peers and asked them to allow us access to their listening history. From there, we sent each of them their own personal playlist along with 10 randomly chosen songs so that we could compare whether our model performed better than pure chance. We asked our volunteers to return a list of 1s and 0s indicating which songs they liked and which they disliked. Now, we compared these real world results with what we saw in testing.

V: Discussion & Result Interpretation

Ultimately, we tested six different machine-learning models and changed various hyper-parameters where applicable. Afterwards, we took the best version of each model and used cross-validation to determine what model is overall going to be the most consistently effective.

Threshold	Train Accuracy	Train Error	Train Precision	Train Recall	Train F1	Train AUC	Test Accuracy	Test Error	Test Precision	Test Recall	Test F1	Test AUC
0.01	0.542953	0.457047	0.522440	1.000000	0.686320	0.542953	0.085960	0.914040	0.004370	1.000000	0.008701	0.541139
0.10	0.656783	0.343217	0.594221	0.988782	0.742330	0.656783	0.322254	0.677746	0.005884	1.000000	0.011699	0.659762
0.25	0.709444	0.290556	0.637131	0.973106	0.770068	0.709444	0.442598	0.557402	0.006810	0.952381	0.013523	0.696463
0.33	0.723562	0.276438	0.654630	0.946453	0.773947	0.723562	0.503916	0.496084	0.007645	0.952381	0.015169	0.727245
0.50	0.746500	0.253500	0.705746	0.845542	0.769345	0.746500	0.658262	0.341738	0.008889	0.761905	0.017573	0.709875
0.67	0.681783	0.318217	0.761265	0.529674	0.624696	0.681783	0.836485	0.163515	0.012835	0.523810	0.025057	0.680777
0.75	0.623130	0.376870	0.795672	0.331352	0.467865	0.623130	0.916523	0.083477	0.014019	0.285714	0.026726	0.602389
0.90	0.502996	0.497004	0.722420	0.009732	0.019204	0.502996	0.992932	0.007068	0.000000	0.000000	0.000000	0.498466
0.99	0.500000	0.500000	1.000000	0.000000	0.000000	0.500000	0.995989	0.004011	1.000000	0.000000	0.000000	0.500000

The first model we considered was logistic regression. We stuck to a max-iteration value of 10,000, and tested different threshold values ranging from 0.01 to 0.99. Increasing the threshold consistently had a positive effect on testing accuracy and a negative effect on training accuracy. The inverse can generally be said with the AUC values. We figured our best model when considering both of these metrics and the tradeoff between them was when we set the decision threshold value to 0.75.

NB Model	Train Accuracy	Train Error	Train Precision	Train Recall	Train F1	Train AUC	Test Accuracy	Test Error	Test Precision	Test Recall	Test F1	Test AUC
GaussianNB	0.745566	0.254434	0.681952	0.920374	0.783425	0.745566	0.571538	0.428462	0.007972	0.857143	0.015796	0.713765

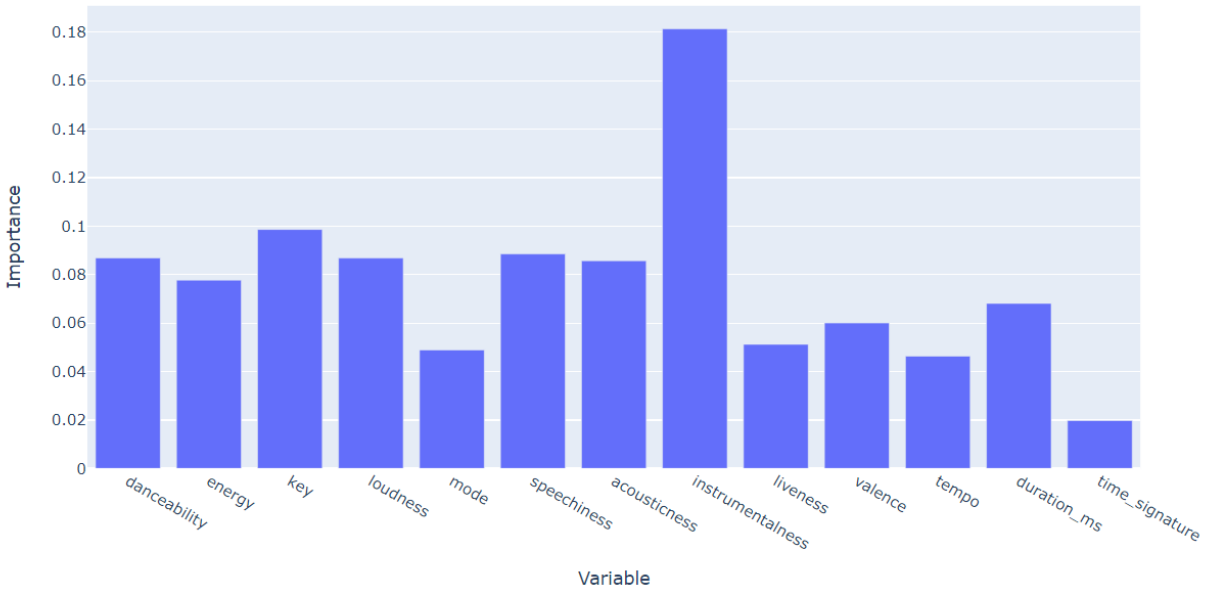
The second model we considered was Naive-Bayes. We did not tune any hyperparameters. Generally speaking, the model underperformed compared to other models, especially in regards to testing accuracy. There was also a noticeably high recall value in testing, as well, which contributed to fairly high AUC.

Criterion	Depth	Train Accuracy	Train Error	Train Precision	Train Recall	Train F1	Train AUC	Test Accuracy	Test Error	Test Precision	Test Recall	Test F1	Test AUC
entropy	1	0.627733	0.372267	0.573219	1.000000	0.728721	0.627733	0.252340	0.747660	0.005337	1.000000	0.010617	0.624664
entropy	5	0.797100	0.202900	0.758649	0.871429	0.811138	0.797100	0.727603	0.272397	0.009085	0.619048	0.017906	0.673544
entropy	10	0.924569	0.075431	0.878434	0.985523	0.928902	0.924569	0.868386	0.131614	0.010264	0.333333	0.019915	0.601937
entropy	20	0.984971	0.015029	0.971258	0.999521	0.985187	0.984971	0.959503	0.040497	0.015228	0.142857	0.027523	0.552825
entropy	50	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.982044	0.017956	0.013333	0.047619	0.020833	0.516713
entropy	100	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.982044	0.017956	0.000000	0.000000	0.000000	0.493000
gini	1	0.628092	0.371908	0.573467	0.999856	0.728883	0.628092	0.253868	0.746132	0.005348	1.000000	0.010638	0.625432
gini	5	0.799233	0.200767	0.764402	0.865101	0.811640	0.799233	0.741165	0.258835	0.009559	0.619048	0.018827	0.680352
gini	10	0.929914	0.070086	0.879550	0.996261	0.934274	0.929914	0.866094	0.133906	0.010086	0.333333	0.019580	0.600786
gini	20	0.978044	0.021956	0.957977	0.999952	0.978515	0.978044	0.945941	0.054059	0.018382	0.238095	0.034130	0.593443
gini	50	0.999521	0.000479	0.999042	1.000000	0.999521	0.999521	0.978415	0.021585	0.010638	0.047619	0.017391	0.514891
gini	100	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.978797	0.021203	0.010870	0.047619	0.017699	0.515083

The third model we considered was the decision tree. We tuned the hyperparameters for splitting criteria and maximum tree depth. Ultimately, we found the model to be effective with greater depths, although we once again were facing relatively low AUC values as a tradeoff for high accuracy. We found the best-performing parameter setup to be the entropy splitting criterion with a maximum depth of 20. This was optimal for its balance of accuracy and AUC, as well as its comparatively decent runtime.

Estimators	Train Accuracy	Train Error	Train Precision	Train Recall	Train F1	Train AUC	Test Accuracy	Test Error	Test Precision	Test Recall	Test F1	Test AUC
10	0.986002	0.013998	0.972810	0.999952	0.986195	0.986002	0.971347	0.028653	0.015038	0.095238	0.025974	0.535057
50	0.986314	0.013686	0.973356	1.000000	0.986498	0.986314	0.972684	0.027316	0.015873	0.095238	0.027211	0.535728
100	0.986337	0.013663	0.973402	1.000000	0.986522	0.986337	0.972493	0.027507	0.023256	0.142857	0.040000	0.559346
500	0.986673	0.013327	0.974038	1.000000	0.986848	0.986673	0.971729	0.028271	0.015267	0.095238	0.026316	0.535249

The fourth model we considered was the random forest. The forest was composed of decision trees tuned with the hyperparameters that we evaluated to be the best-performing. For the hyperparameters of the forest we considered different amounts of estimators. Generally speaking, all of the models performed similarly regardless of the number of estimators, with the only notable difference being that more estimators yielded higher recall.



Using the random forest model, we also chose to look at feature importance within our data. For this given user, the most important feature was instrumentalness. This intuitively makes sense since different genres are often distinguished between one another through the use of instruments. Key was also a variable of large importance, which is usually a good indicator of a song's mood. Time signature was the least important variable, which makes sense since many songs already are in the same time signature to begin with.

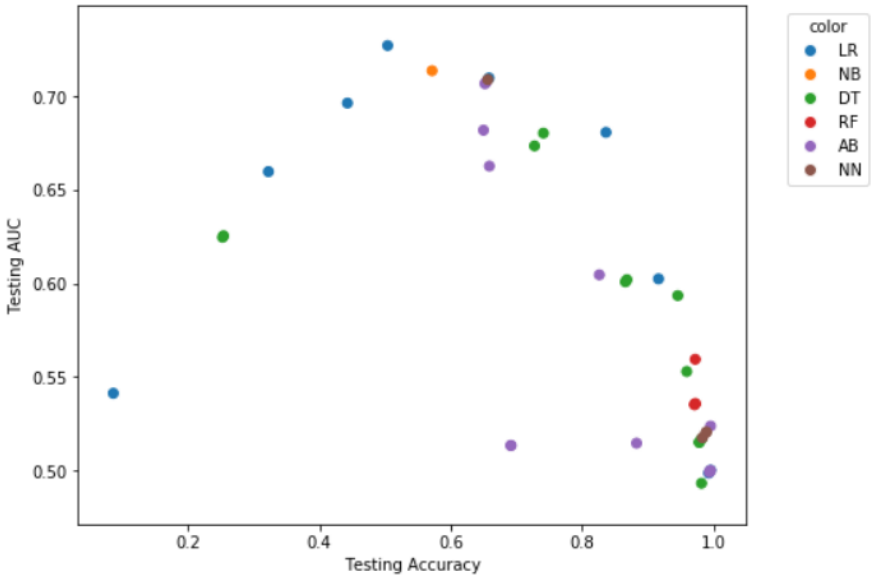
Base Estimator	Classifiers	Train Accuracy	Train Error	Train Precision	Train Recall	Train F1	Train AUC	Test Accuracy	Test Error	Test Precision	Test Recall	Test F1	Test AUC
LogisticRegression	10	0.746213	0.253787	0.707969	0.838159	0.767583	0.746213	0.658835	0.341165	0.007808	0.666667	0.015436	0.662735
LogisticRegression	50	0.741898	0.258102	0.701276	0.842809	0.765556	0.741898	0.649857	0.350143	0.008143	0.714286	0.016103	0.681941
LogisticRegression	100	0.741012	0.258988	0.700371	0.842426	0.764858	0.741012	0.653868	0.346132	0.008777	0.761905	0.017354	0.707669
LogisticRegression	500	0.740772	0.259228	0.699996	0.842713	0.764753	0.740772	0.652149	0.347851	0.008734	0.761905	0.017269	0.706806
GaussianNB	10	0.533054	0.466946	0.579665	0.240508	0.339963	0.533054	0.826361	0.173639	0.008850	0.380952	0.017297	0.604554
GaussianNB	50	0.517306	0.482694	0.562958	0.154746	0.242762	0.517306	0.883095	0.116905	0.005025	0.142857	0.009709	0.514467
GaussianNB	100	0.537248	0.462752	0.552664	0.390892	0.457910	0.537248	0.691691	0.308309	0.004356	0.333333	0.008600	0.513234
GaussianNB	500	0.537248	0.462752	0.552664	0.390892	0.457910	0.537248	0.691691	0.308309	0.004356	0.333333	0.008600	0.513234
DecisionTreeClassifier	10	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.993887	0.006113	0.000000	0.000000	0.000000	0.498945
DecisionTreeClassifier	50	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.995415	0.004585	0.000000	0.000000	0.000000	0.499712
DecisionTreeClassifier	100	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.995798	0.004202	0.333333	0.047619	0.083333	0.523618
DecisionTreeClassifier	500	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.995606	0.004394	0.000000	0.000000	0.000000	0.499808

The fifth model we considered was AdaBoost. We trained AdaBoost models using various different base estimators and amounts of classifiers. With our decision tree models we found there to be no level of precision or recall. Our Naive Bayes model performed fairly well when

there were 50 or less classifiers. Generally speaking, the AdaBoost results were inconsistent and very dependent on the hyperparameters set.

Activation	Train Accuracy	Train Error	Train Precision	Train Recall	Train F1	Train AUC	Test Accuracy	Test Error	Test Precision	Test Recall	Test F1	Test AUC
identity	0.744487	0.255513	0.703463	0.845302	0.767887	0.744487	0.656543	0.343457	0.008845	0.761905	0.017486	0.709011
logistic	0.996021	0.003979	0.992105	1.000000	0.996037	0.996021	0.983381	0.016619	0.014706	0.047619	0.022472	0.517385
tanh	0.999545	0.000455	0.999090	1.000000	0.999545	0.999545	0.988921	0.011079	0.025641	0.047619	0.033333	0.520165
relu	0.999736	0.000264	0.999473	1.000000	0.999736	0.999736	0.989494	0.010506	0.027778	0.047619	0.035088	0.520453

The sixth and final model we considered was the neural network MLP classifier. We considered using different activation functions. Once again, we found trouble finding a balance of AUC and accuracy. It is also worth noting that in most instances our models did not converge before reaching the maximum number of allotted iterations.



When looking at how the models performed in terms of testing accuracy and AUC, we see many models with extremely high accuracy and low AUC. Given the skew in target classes in our dataset, this is likely due to the most accurate models being the ones that predict very few positives. However, we see some instances of decision trees and logistic regression that find a middle-ground of both metrics.

model	accuracy	error	precision	recall	f1	auc
LogisticRegression	0.738993	0.261007	0.695051	0.851653	0.765415	0.804864
GaussianNB	0.746778	0.253222	0.682228	0.923947	0.784895	0.864373
DecisionTreeClassifier	0.969510	0.030490	0.949373	0.991409	0.969949	0.979554
RandomForestClassifier	0.987094	0.012906	0.976513	0.999616	0.988077	0.999670
AdaBoostClassifier	0.997200	0.002800	0.995452	0.998926	0.996709	0.999855
MLPClassifier	0.993308	0.006692	0.986980	1.000000	0.992879	0.998721

As a final test, we took the best-performing iteration of each model we implemented and compared their performance using cross-validation. With cross-validation, we actually saw significant improvement in every model. Particularly, the decision trees, random forests, AdaBoost (with a decision tree base), and the neural network all had both an accuracy and AUC above 0.98. When testing on real-world volunteers, we opted to use the AdaBoost implementation.

After receiving feedback from our volunteers, we concluded that our model and dataset had a few issues. Many people reported not seeing a great difference between the 10 recommended songs and the ten randomly selected songs. We believe that this comes down to a few things. First, when using a Spotify API call, only 50 songs can be retrieved. We attempted to improve this issue by using two API calls, getting both saved songs and most listened to songs. However, we believe that our model would benefit from having more training data as there are many different features that need to be considered.

On the other hand, our unknown dataset was extremely large and unfiltered. When examining real world models of this project, such as Spotify's Discover Weekly playlist, it is clear that the dataset of songs that is chosen from is tailored to the users most listened to genres and artists. This was difficult to achieve in this project as we did have access to some of this user data or some of these datasets. Choosing from such a large and diverse dataset proved challenging as many songs can have similar audio features but be completely different genres. So even though the acousticness and danceability may match up with what the user typically enjoys, it does imply that they will enjoy that particular song.

If we continued to work on this project, we would have most likely chosen a dataset based on the user's individual taste. For example, if we knew the user enjoyed Rock the most, then we could scrape songs from Spotify's curated Rock playlists or popular Rock artists. This would increase the accuracy of our predictions as we would be narrowing our search much more than we have in this project.

VI: Conclusion

One of the most interesting observations made during the testing of these machine-learning projects was the discrepancy between our accuracies reported in model testing and real-world testing. Despite some of our models performing with 99% accuracy in cross-validation testing, we found that using the same model to recommend songs to real people yielded fairly unimpressive results more often than not. For the most part, our subjects did not feel that the songs we were recommending were more enjoyable for them than songs we selected at random. This could mean that there is something to say about the data we are working with; there likely are features we do not have access to that have a strong effect on how people perceive different music. This would account for the discrepancy between theoretical testing and real-life testing.

Another point of interest is that our model has persistently low precision and recall values, particularly during testing. This can likely be explained by the differences in class size; even with the implementation of SMOTE we are really only working with up to 100 unique songs in the positive class. Logically speaking, it makes sense that based on this we would overall not see too many predicted positives.

Generally speaking, this project allowed us to explore various machine learning models and principles in an area of interest and relevance. We were able to work on a compelling problem with an interesting dataset and tested six different machine learning models with different hyperparameters at play. While song predication may not be as simple as we had hoped, this project allowed us to have some interesting insights into music, the music industry, and how music streaming platforms recommend songs curated to the tastes of each individual user.

VII: Contribution

Work was split mostly 50/50 between partners. Code-work was done in multiple paired-programming sessions, and the writing of the paper was split evenly.