Playlist Creation Using Audio Features

Predicting Playlist Genre With Spotify Audio Data

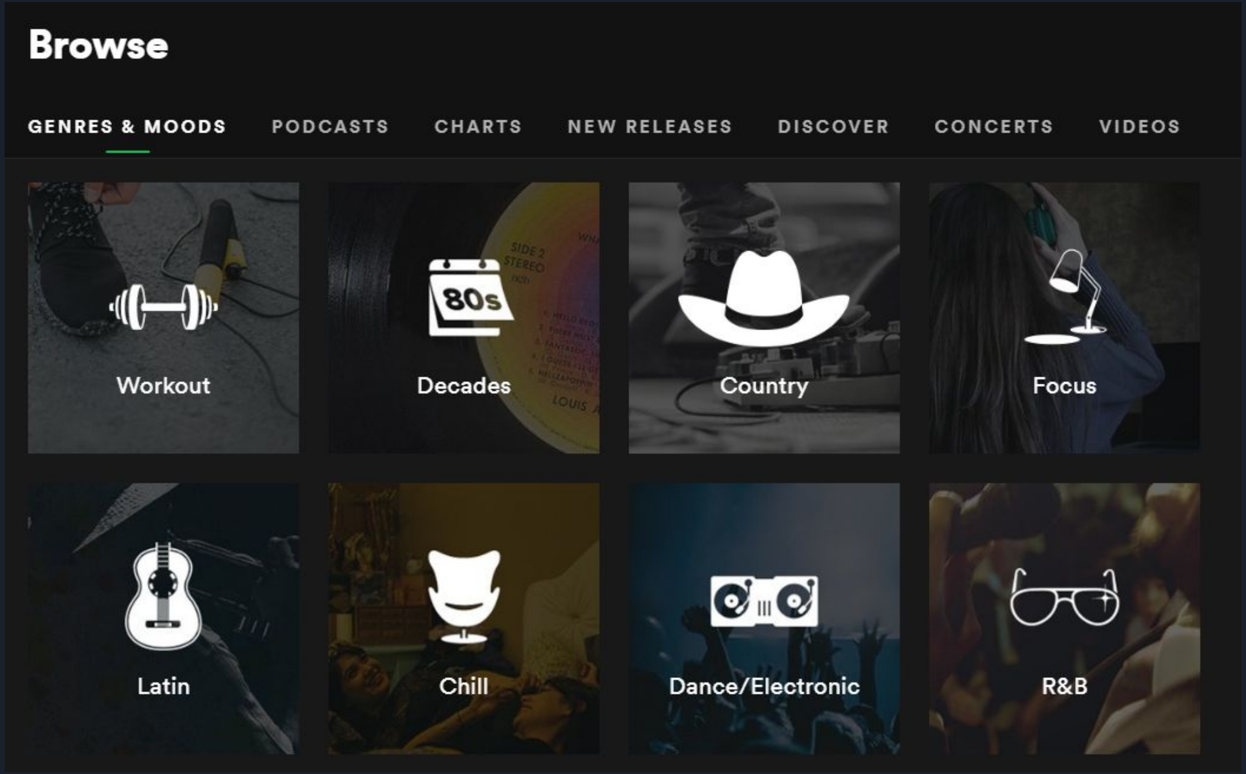
Amanda Strack

Background and Motivation

- Different songs can set different moods. Would you relax to the same music you would workout to?
- Many people have different playlists based on a certain activity or mood.
- However, it can be time consuming to create these playlists. It requires sifting through many songs and then determining if it fits a certain category based on how it sounds and makes you feel.
- My goal is to use machine learning to classify songs into different playlists.
- I will do this by using certain audio features of a song to classify which playlist category the song best fits.
- My client could be any music streaming company such as Spotify. The client will be able to use this data-driven product feature to enable users to more easily discover songs through pre-made playlists.

The Client

- Any music streaming company will be able to use this data-driven product feature to enable users to more easily discover songs through pre-made playlists
- User can create a playlist by selecting a category or activity



The Data

- The data contains 13 audio feature metrics of a song such as ‘Tempo’ or ‘Instrumentalness’.
- Data was extracted from the Spotify API. Songs were extracted from pre-existing playlists on the Spotify platform.
- Our target variable is the playlist category:
    - Workout
    - Party
    - Chill
    - Focus

Which Playlist??

Workout

Focus

Chill

Party

How will it work?

Models

Audio feature data is used to build models that predict which playlist genre a song belongs to.

Make Predictions

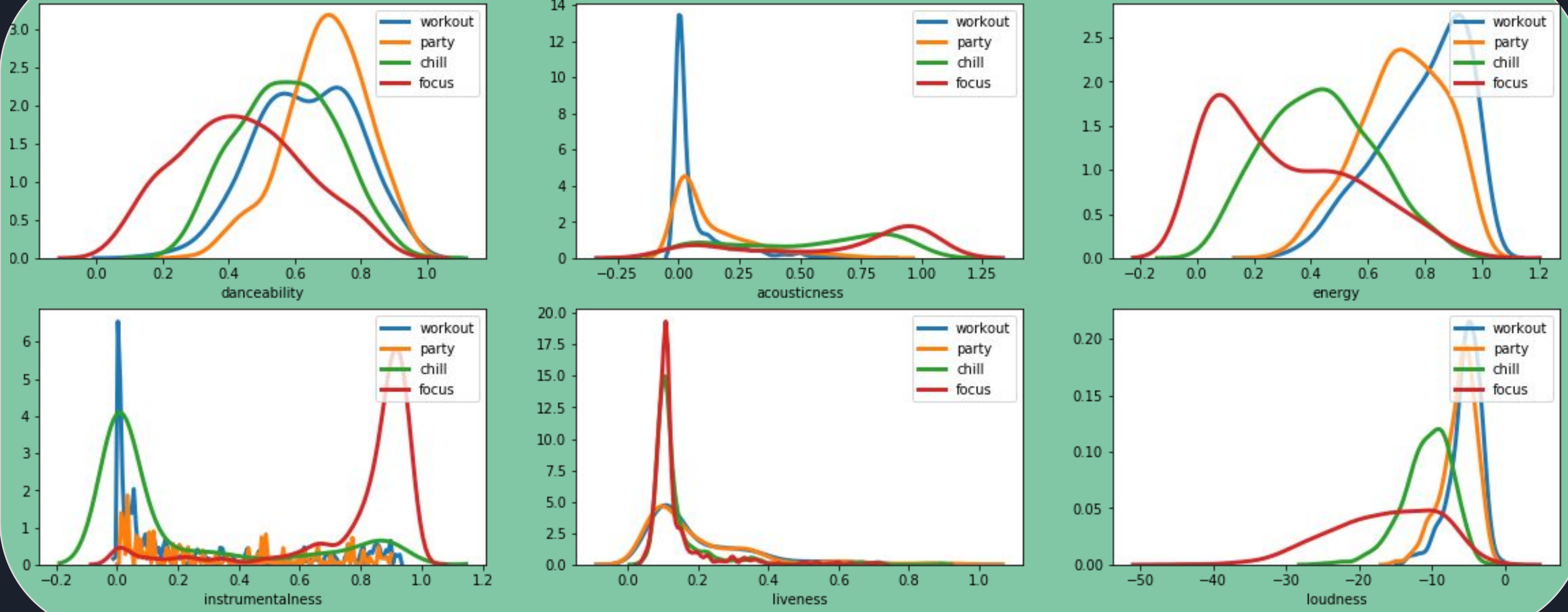
Songs are fed into the model and labeled a genre or activity.

Create Playlists

Songs are added to their respective genres resulting in the creation playlists.

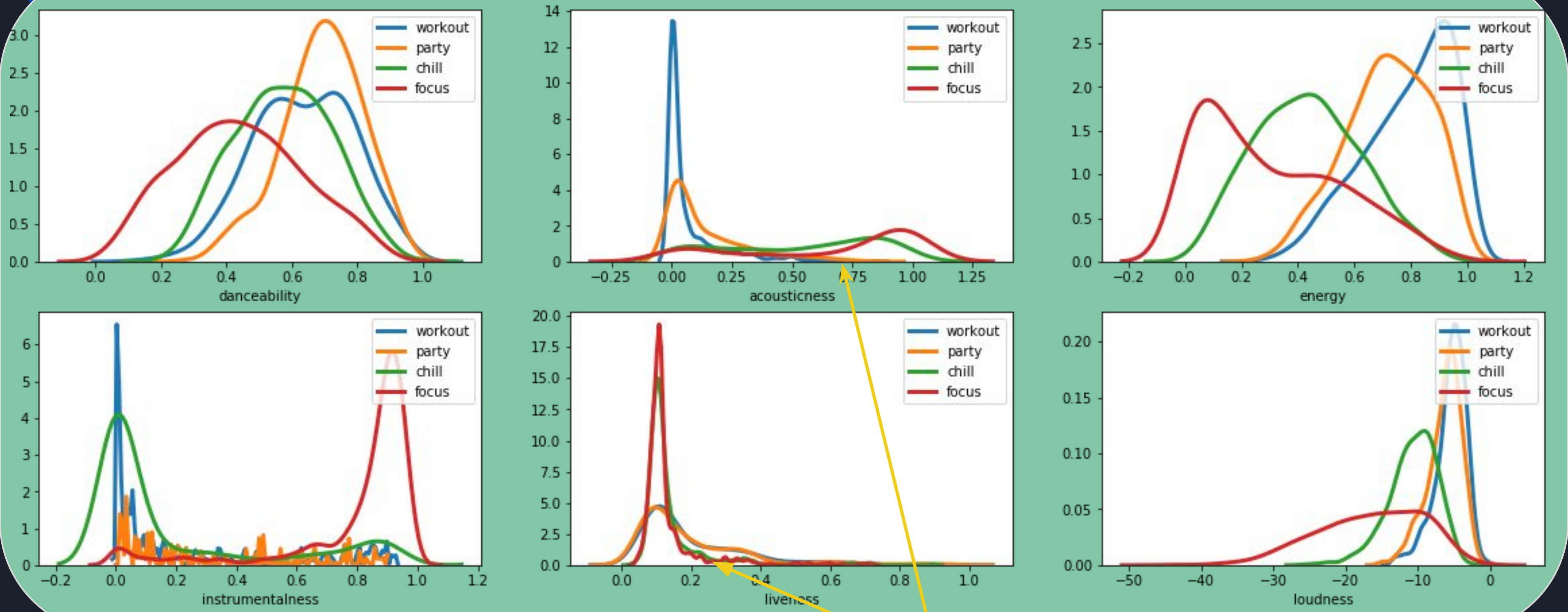
EDA

Are the variables ‘different’ enough?



A few KDE plots of the audio features of the different playlists.

EDA Are the variables ‘different’ enough?



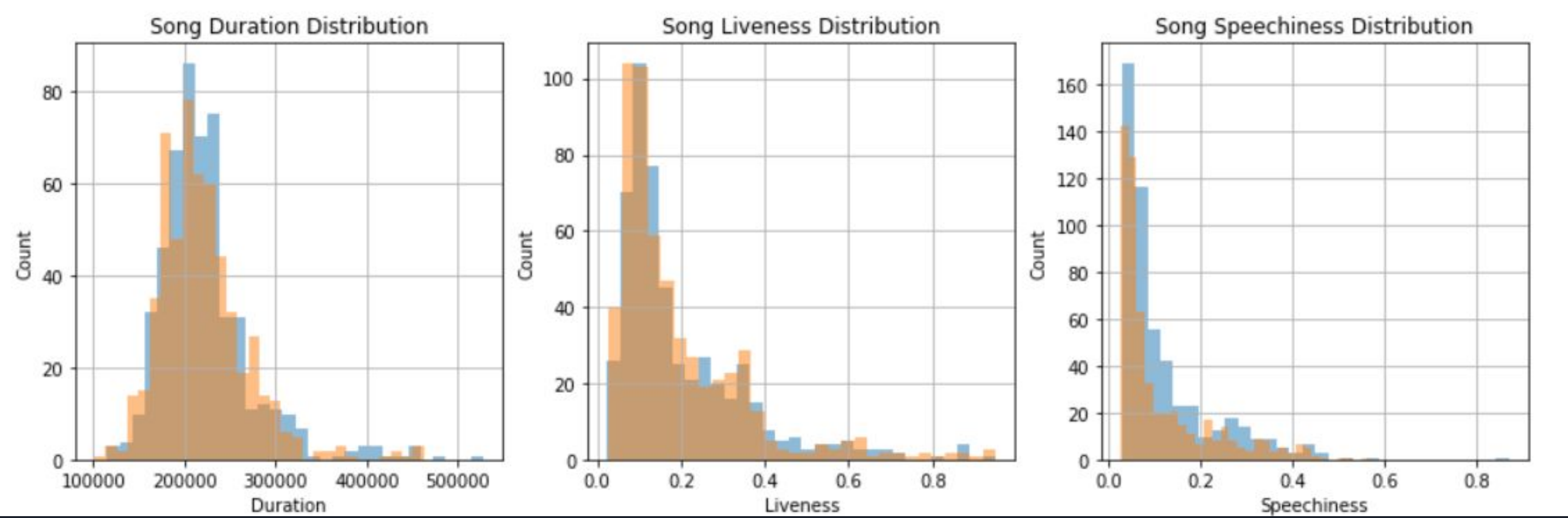
It is difficult to distinguish playlists in some graphs.

Are the variables ‘different’ enough?

- We use hypothesis testing to see if the distributions of the features are significantly different across playlist genres
- Results will give us an idea of the features that will be strong predictors in our model
- Statistical tests used:
  - T-test, Chi-Squared test for Independence, and Kruskal-Wallis

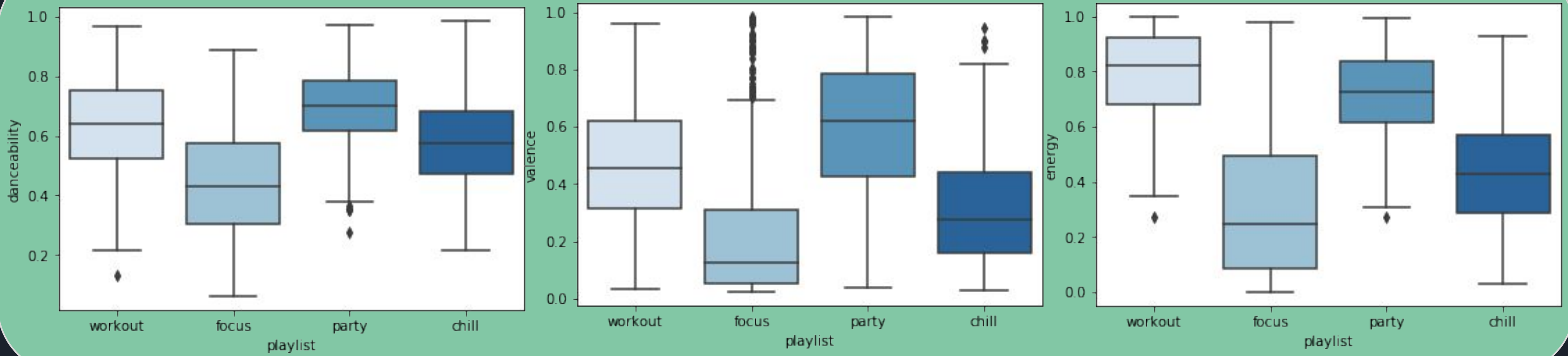
Workout vs. Party...Too Similar?

- Workout and Party playlists had several overlapping songs in our initial extraction of the data
- Prompted me to test if they were significantly different enough from each other
- **Liveness, Duration, Speechiness, Mode, Time Signature**, and **Key** variables tested to have **no** significant difference in distributions between Workout and Party playlists
- This may cause problems with our model! (keep this in mind)



What about across all playlist genres?

All variables tested to have significantly different distributions when testing across all four genres. These boxplots help visualize these differences.



Seems like our variables will be pretty strong predictors...now let's get to modeling!

Models

01	**Random Forest**	• Predict the label of a data point by ensembling decision trees
02	**Support Vector Machines (SVM)**	• Defines a hyperplane that separates classes
03	**Naive Bayes**	• Makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting
04	**K-Nearest Neighbors (KNN)**	• Predict the label of a data point by looking at the ‘K’ closest labeled data points and taking the majority vote
05	**XGBoost**	• Implements gradient boosted decision trees

Results

We began by implementing the models with their default parameters. Here are the results from the test data.

Model	Accuracy Score	F1 Score
Random Forest	69.81%	.70
SVM	71.39%	.72
Naive Bayes	67.38%	.68
KNN	66.95%	.68
XGBoost	72.53%	.73

Lets focus on XGBoost since had the best performance. Next is hyperparameter tuning….

Hyperparameter Tuning: XGBoost

Tuned Hyperparameters: max_depth, min_child_weight, gamma, subsample, colsample_bytree, learning_rate, n_estimators

Model	Accuracy Score	F1 Score
Tuned XGBoost	73.25%	.73

- The accuracy score has improved.
- The weighted F-score is the same as our initial model.

Hyperparameter Tuning: XGBoost

Base Model

Classification Report:

	precision	recall	f1-score	support
0	0.60	0.59	0.59	181
1	0.83	0.80	0.81	178
2	0.59	0.63	0.61	171
3	0.91	0.88	0.89	169
micro avg	0.73	0.73	0.73	699
macro avg	0.73	0.73	0.73	699
weighted avg	0.73	0.73	0.73	699

Accuracy: 0.7253218884120172

Tuned Model

Classification Report:

	precision	recall	f1-score	support
0	0.62	0.57	0.59	181
1	0.85	0.80	0.82	178
2	0.59	0.67	0.63	171
3	0.90	0.89	0.90	169
micro avg	0.73	0.73	0.73	699
macro avg	0.74	0.73	0.74	699
weighted avg	0.74	0.73	0.73	699

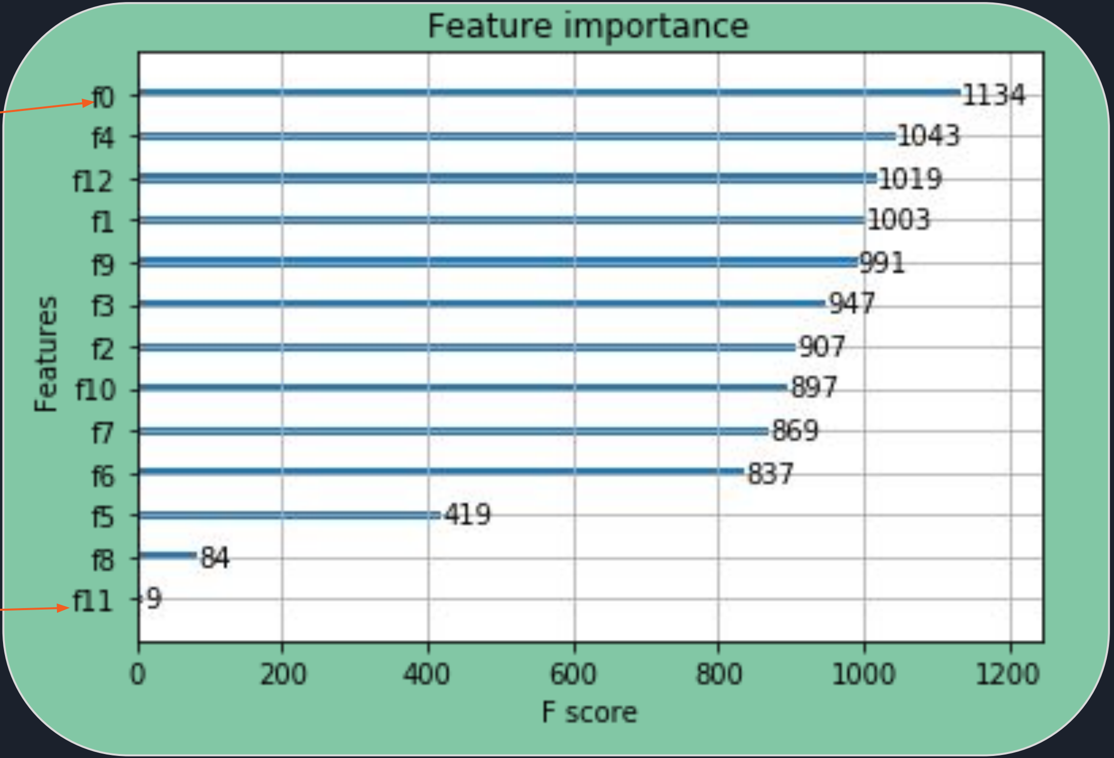
Accuracy: 0.7324749642346209

- The accuracy score has improved.
- The weighted F-score is the same as our initial model.

Feature Importance

acousticness

time signature

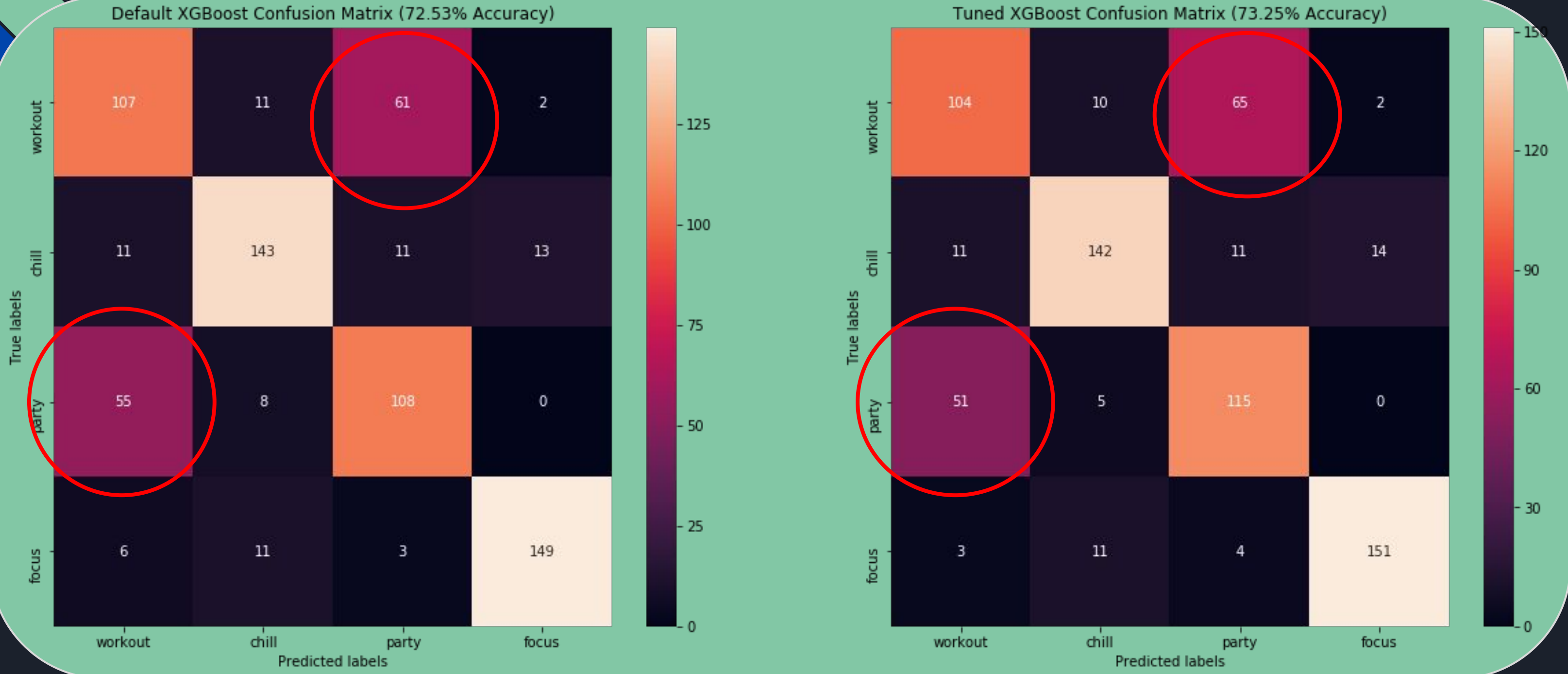


Feature Thresholds

- Test the model performance at each threshold of features by importance
- Score is from accuracy of the training set
- Accuracy is generally decreasing as the number of features go down
- Using all the features gives us the best score, therefore we won't remove any features in our final model

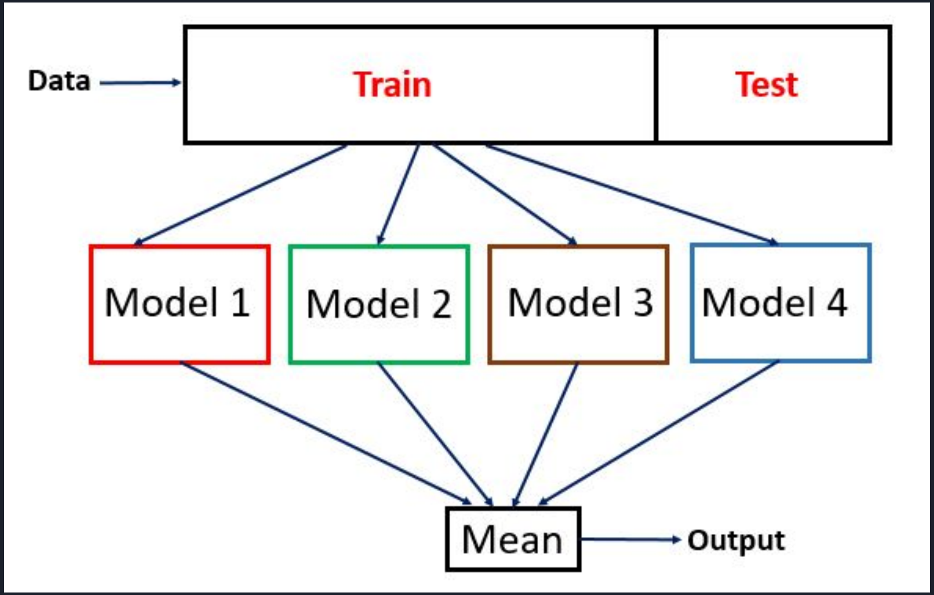
```
Thresh=0.0078, n=13, Accuracy: 73.25%
Thresh=0.0364, n=12, Accuracy: 72.82%
Thresh=0.0397, n=11, Accuracy: 72.53%
Thresh=0.0451, n=10, Accuracy: 73.10%
Thresh=0.0468, n=9, Accuracy: 71.96%
Thresh=0.0562, n=8, Accuracy: 71.82%
Thresh=0.0599, n=7, Accuracy: 71.10%
Thresh=0.0696, n=6, Accuracy: 68.81%
Thresh=0.0845, n=5, Accuracy: 70.82%
Thresh=0.0996, n=4, Accuracy: 63.81%
Thresh=0.1088, n=3, Accuracy: 64.66%
Thresh=0.1142, n=2, Accuracy: 59.94%
Thresh=0.2315, n=1, Accuracy: 48.35%
```

Final Tuned XGBoost Model



The model often 'confuses' workout and party classes. We expected this from our earlier exploration of the data. It is the confusion of these two classes that is preventing us from achieving a more accurate model.

Ensembling



- Since our model is having difficulty classifying between workout and party songs, we will use an ensemble method.
- Ensembling refers to the method of combining several machine learning techniques in order to improve predictions.
- Train two more models on only class 0 (workout) and class 2 (party):
  - Random Forest
  - SVM
- We average predicted probabilities of the ensemble of models to get our final predictions.

Ensembling

After hyperparameter tuning....

Classification Report:

	precision	recall	f1-score
0	0.67	0.64	0.65
2	0.63	0.67	0.65
micro avg	0.65	0.65	0.65
macro avg	0.65	0.65	0.65
weighted avg	0.65	0.65	0.65

Accuracy: 0.6505681818181818

Random Forest

Classification Report:

	precision	recall	f1-score
0	0.67	0.61	0.64
2	0.62	0.68	0.65
micro avg	0.64	0.64	0.64
macro avg	0.64	0.64	0.64
weighted avg	0.64	0.64	0.64

Accuracy: 0.6420454545454546

SVM

Now let's combine all three models...

Final Ensemble Model

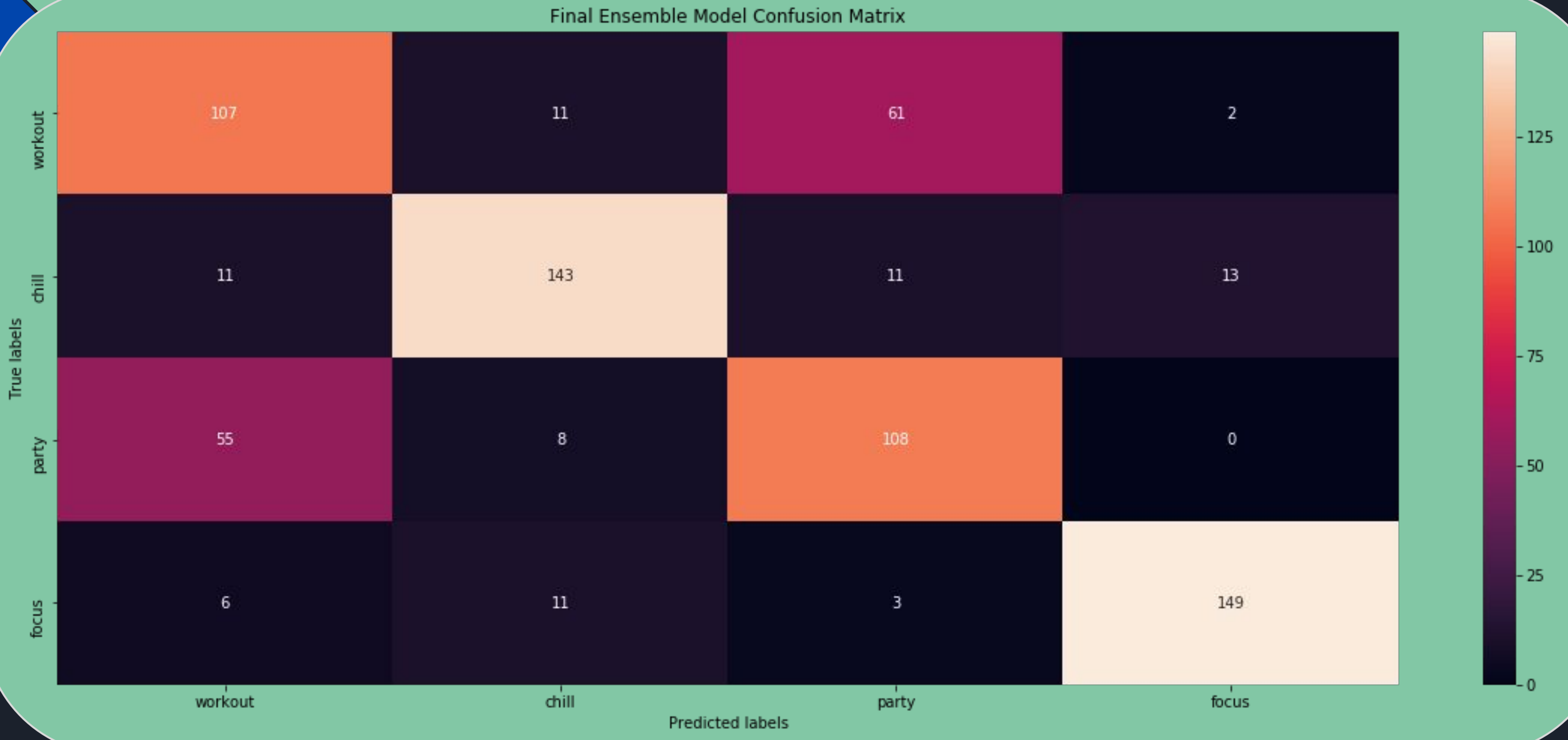
When we combine our models, we see an improvement of 1.14% accuracy and .02 f1-score from our tuned XGBoost model.

XGBoost

Random Forest + SVM

Model	Accuracy Score	F1 Score
Ensemble	74.53%	.75

Final Ensemble Model



Recommendations

- Although our ensemble improved our model, we still see most of our error coming from the confusion of workout and party
- To further improve the model, more features must be added to more clearly distinguish party and workout songs
- Include more audio metrics extracted from the *Librosa* Python package
- Add in more metadata
  - Song Artists
  - Song Popularity
- Personalized data using Spotify user data
  - Listening history
  - Favorite artists

Next Steps

- Include several more playlist categories to classify
- Create a platform that allows a user to automatically create a playlist of songs based on their selection of activity or mood