Music Recommendation System

Using Matrix Factorization and Spotify Audio Data

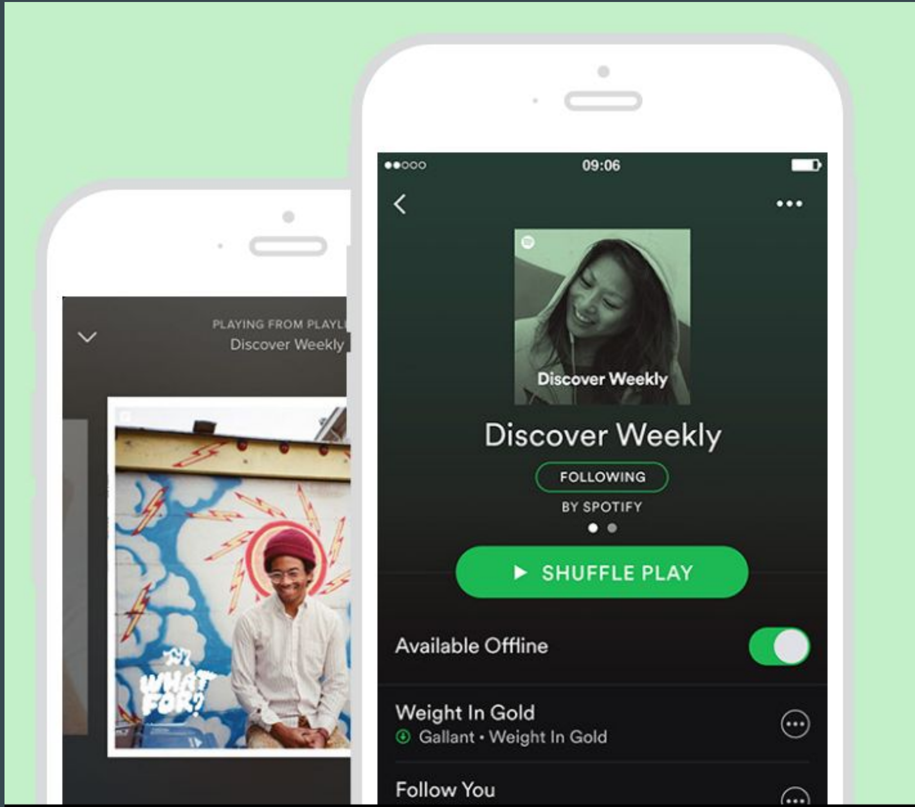
Amanda Strack

Background and Motivation

- Recommendation systems are vitals features for companies such as Amazon, Netflix and Google.
- The is to personalize content and to present the most relevant items to the customer. This creates a better user experience while also driving revenue.
- In this project I will build a recommendation system for users to discover new music.

The Client

- My client is 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 that they are likely to enjoy.
- This feature will ultimately benefit the company because it will enhance user experience and keep users engaged.



The Data

- The first part of my dataset comes from the Million Song Dataset. The data contains user_id, song_id and listen count, title, release_by (release date) and artist_name. There are 10,000 unique songs.
- The next part of the dataset was called from the Spotify Web API. I exported a CSV of the 10,000 song names from our previous dataset and created Spotify playlists of ~7,000 of those songs. Using the Spotify API, I extracted the artist name, song title, and spotify ID of the tracks of these playlists. I then merged this dataset with our previous dataset.
- The last part of the dataset are audio features also called from the Spotify API. The audio features were extracted using the spotify song ID's of our dataset and then were added in as additional columns.

How will it work?

Matrix Factorization

Matrix factorization is used to discover latent features underlying the interactions between users and songs.

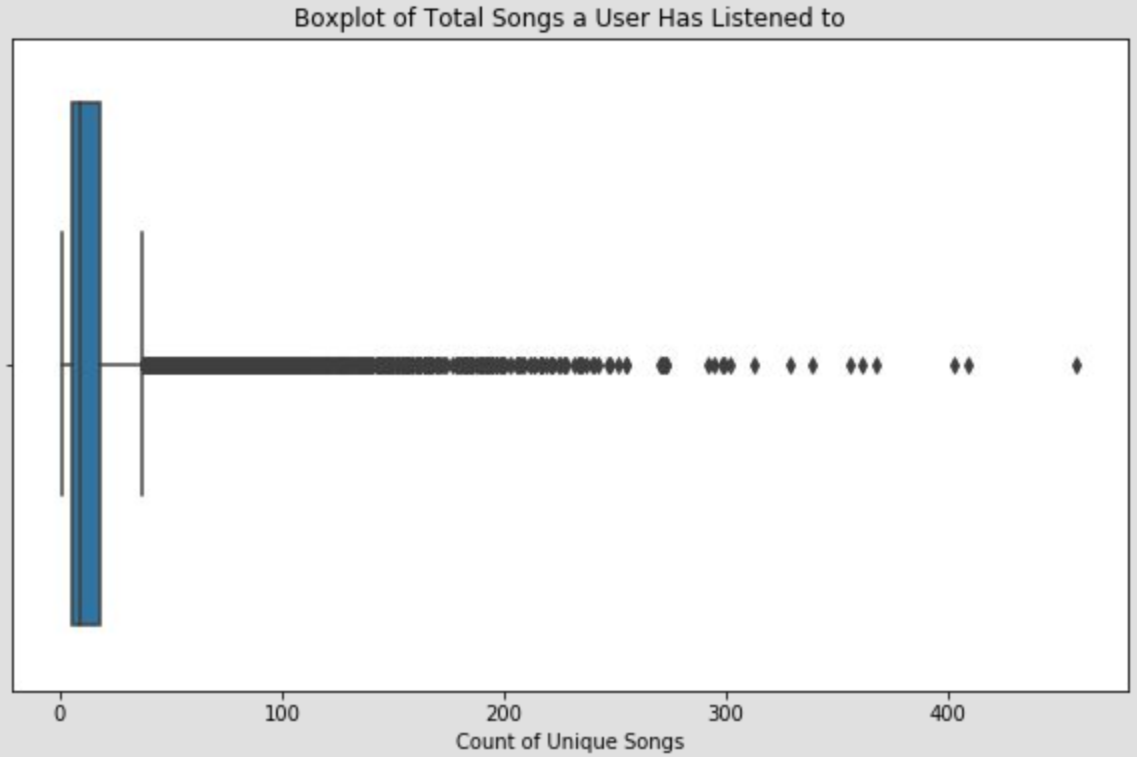
Classification

Once the latent features are joined with the spotify audio features, classification is performed to predict the listen count of song.

Make Recommendations

The outcome of the classification model is used to recommend a user top n songs.

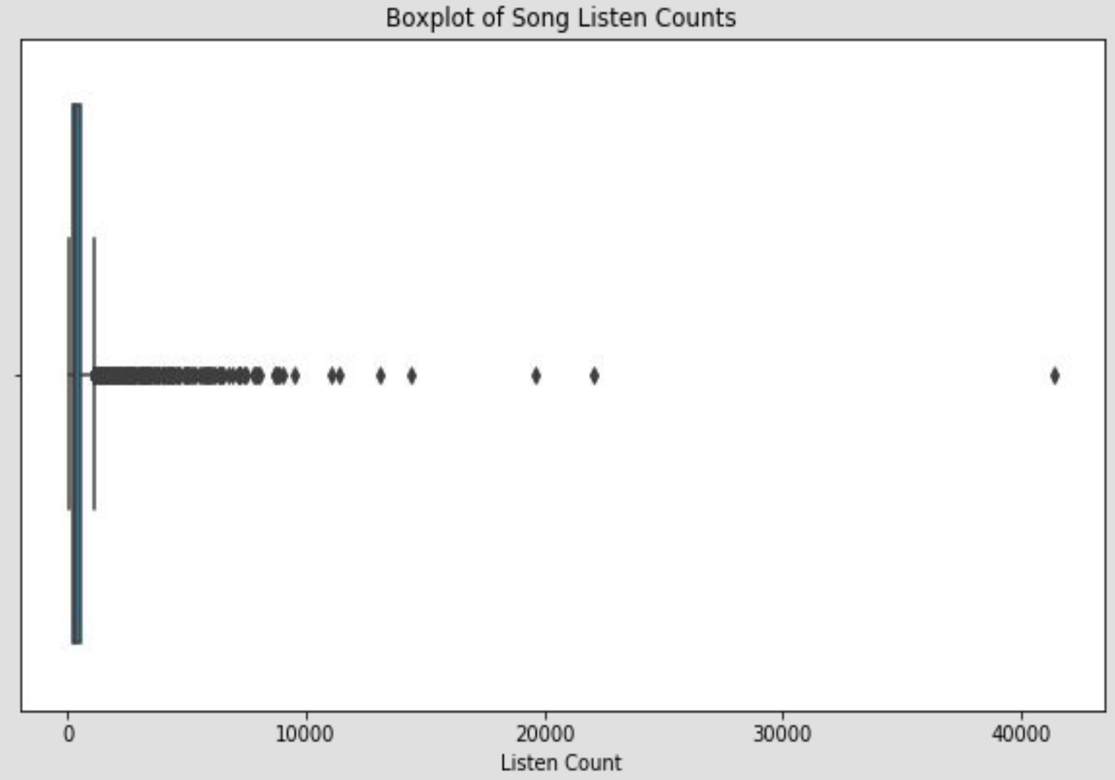
How many different songs does a user actually listen to?



How many different songs does a user actually listen to?

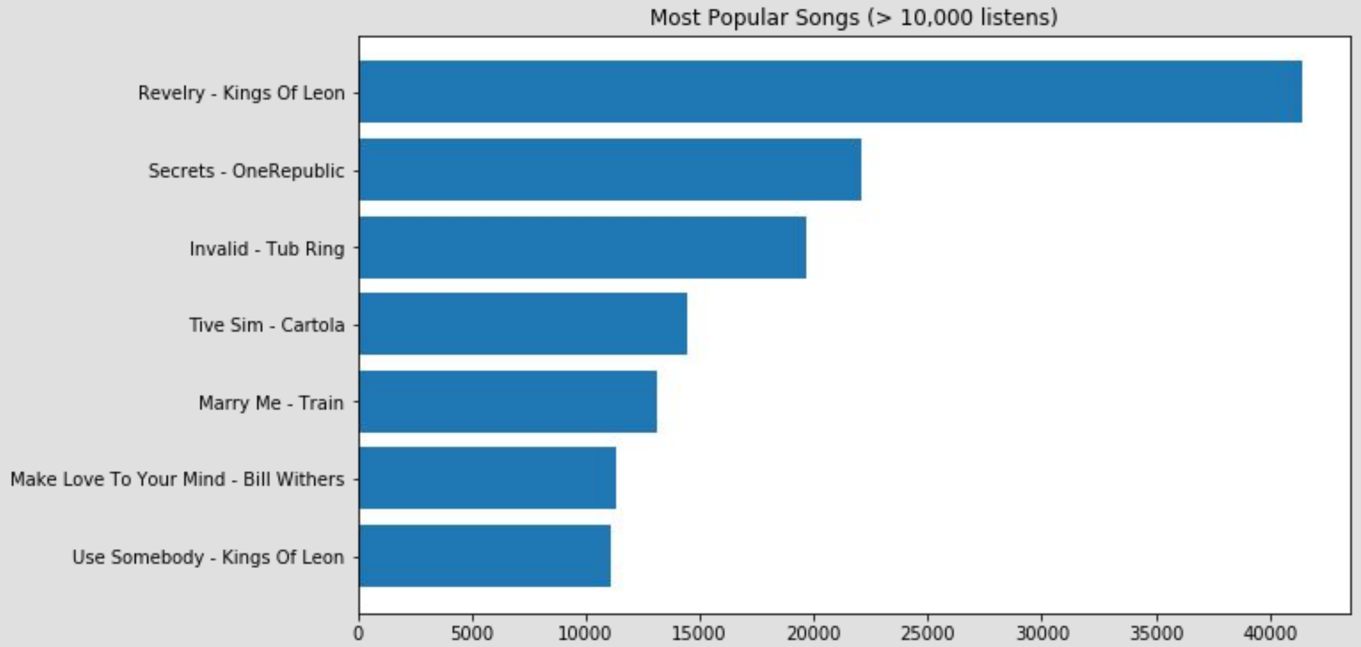
- We want to see how many different songs each user listens to. We want make sure that our data makes sense with real life user listening habits and to also make sure the data is valid enough to train our recommendation machine.
- There are quite a few outliers in regards to the amount of songs a user listens to but since we are building a recommendation system, outliers are treated differently as it is valid data of human activity.
- What we are concerned about is having a dataset that would not allow us to train a recommendation system. In our data, only 3,357 out of 74,904 users only listened to one song (5%) so we can conclude that our data is valid for our model.

How many times is each song listened to?



How many times is each song listened to?

We don’t want a song to be recommended to users only because it's listen count is considerably higher than all other songs. The songs below would be considered outliers but we do not want to remove them from our data. But we do want to keep these songs in mind during our modeling stage so that they will not be recommended to users solely due to their popularity.



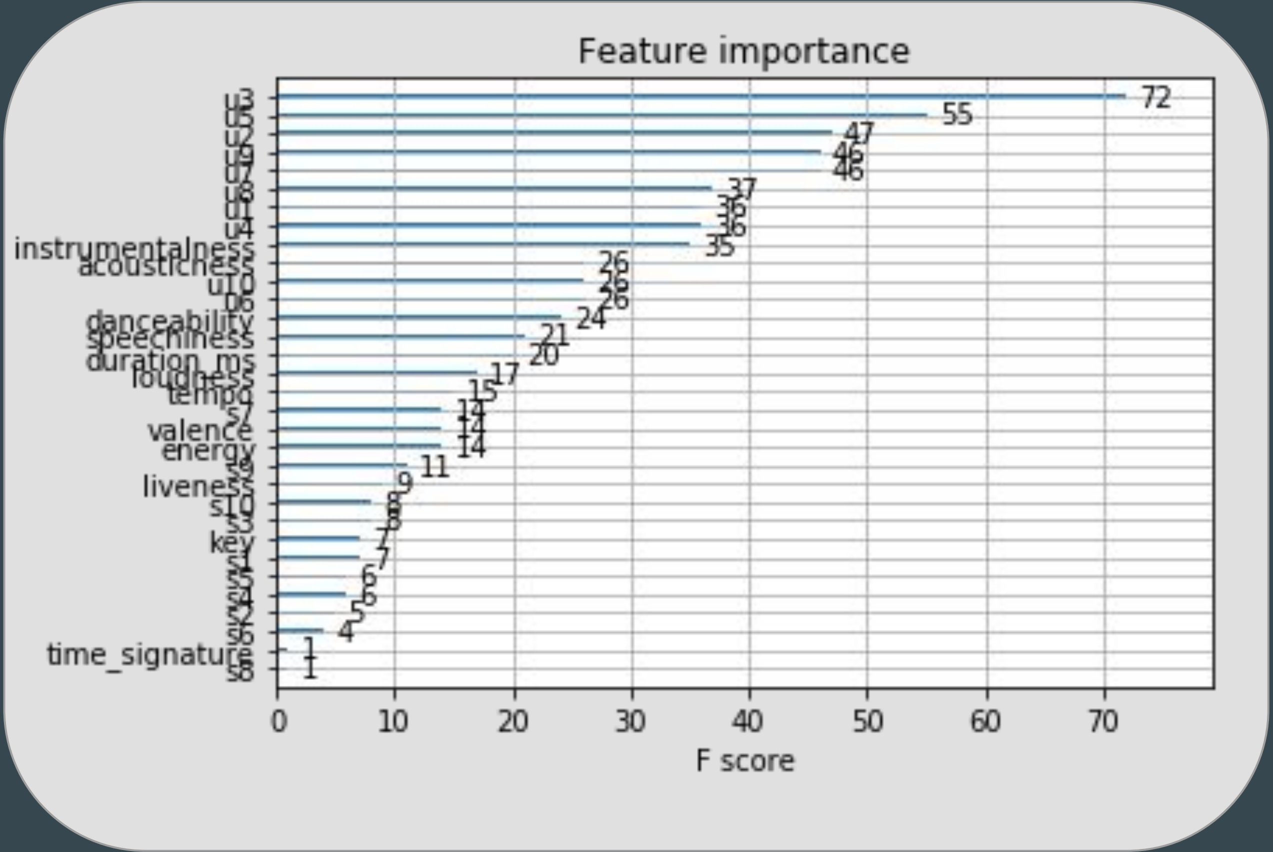
Seems like we have a valid dataset for a recommendation system...now let's get to modeling!

Steps

01	**Matrix Factorization**	• Extract latent features
02	**XGBoost**	• Train XGBoost with latent and audio features.
03	**Random Forest**	• Train Random Forest on same dataset.
04	**Ensembling**	• Ensemble the classification models by averaging predictions of XGBoost and Random Forest.

Feature Importance

- The user latent factors have very high ‘importance’ in comparison to our other while our song or item latent factors have low ‘importance’.
- We found that dropping the 9 features with the lowest F scores ('mode') gives us the optimal AUC score.
- Most of the dropped features were the song latent features.

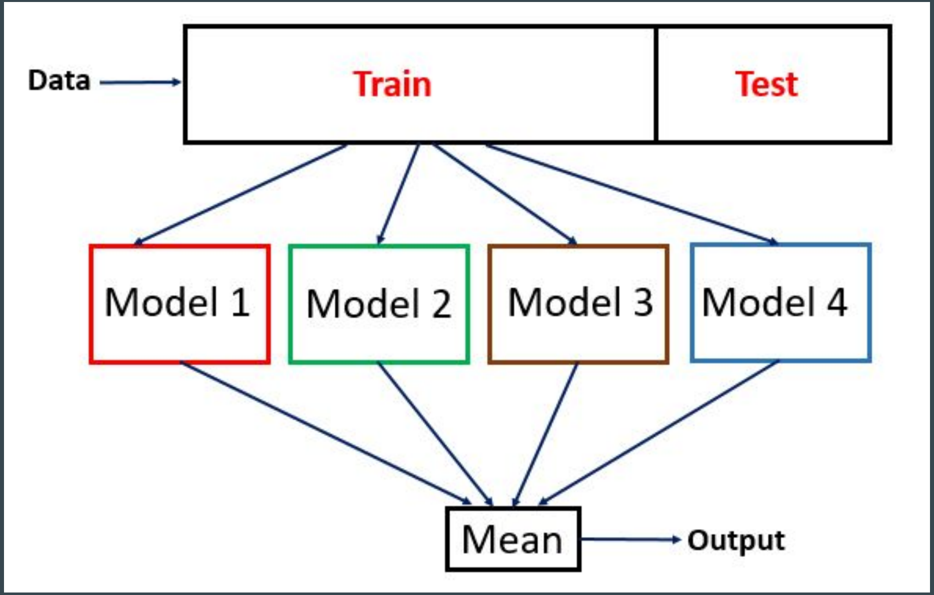


Results

Turns out removing 9 of the least important features gives us an optimal AUC! After we removed those features we then tuned the classification models. The table below shows the AUC before and after hyperparameter tuning and feature selection.

Model	AUC (before)	AUC (after)
XGBoost	.60	.66
Random Forest	.62	.67

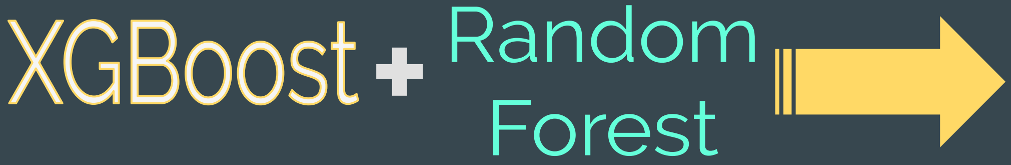
Ensembling



- We use an ensemble method to increase AUC.
- Ensembling refers to the method of combining several machine learning techniques in order to improve predictions.
- We average predicted probabilities of the ensemble of models to get our final predictions.

Final Ensemble Model

When we combine our models, we see an improvement of .08 AUC from our base XGBoost model.



Model	AUC Score
Ensemble	.68

Recommendations

- This type of model requires a ton of data. The number of user and item pairs is very large so we are training on a very small subset of the universe of possibilities.
- More data is necessary for a better score. It is difficult to create a good recommender system with a small amount of data.