Predicting Song Genre Using PCA and XGBoost

Ivan Xiong

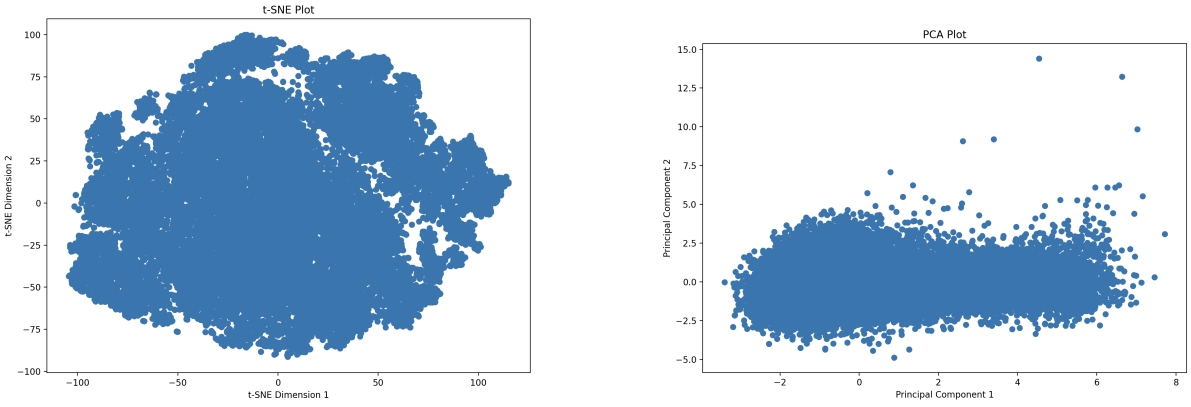
May 28, 2024

The goal of our classifier is to predict the genre of a song given certain characteristics about the song. We had the following predictors: *instance_id*, *artist_name*, *track_name*, *popularity*, *acousticness*, *danceability*, *duration_ms*, *energy*, *instrumentalness*, *liveness*, *loudness*, *speechiness*, *tempo*, *obtained_date* and *valence*. Following are the steps taken regarding data cleaning, training/testing set selection, dimensionality reduction, clustering, and a XGBoost classifier.

The first design choice that I made was regarding the missing data. There were some entries that were missing the *duration_ms* column, and it was replaced with -1. I computed the average duration of the songs (that did not have duration -1) and assigned missing values to that average. Since the missing data is a small subset of the entire dataset, it should not have a large impact on our predictions. The next group of missing data was in the *tempo* column. It was replaced with a "?". Unlike the duration, utilizing domain knowledge we can come to the conclusion that tempo can depend on many other factors. Notably, in music tempo can be correlated with *acousticness*, *danceability*, *energy*, *loudness* among other predictors. However, for the sake of data cleaning, I stuck with these 4. I fit a linear regression model and used the prediction to fill in the *tempo* column if it was missing. I also changed mode from major/minor to 1/0. Any other column that needed to be was transformed from String to numerical data.

To prevent leakage and ensure an equal classification of all genres, we can strategically perform our train test split in this manner: randomly sample 500 songs from each genre for the testing set and randomly sample 4500 songs from each genre for the training set. I did this by first randomly sampling 500 songs for the testing set and putting it in its own dataframe. Next, I dropped those 500 songs from the original data, and then resampled for the 4500 songs for the training set. Since the testing set data was dropped from the dataset before resampling, we ensure that no leakage occurs.

The next part was the dimensionality reduction. We first have to normalize the data, especially for PCA (otherwise PCA will latch onto the largest magnitude scale). I first performed a dimensionality reduction into 2 dimensions using t-SNE to visualize the data and clusters. However, I chose not to stick with the t-SNE for the rest of the classification because in order to maintain the global structure of the data as well as the variance, we should use PCA. Additionally, PCA does not assume a gaussian distribution (since our data might not follow a gaussian distribution). PCA is also computationally inexpensive and can be applied to our relatively large dataset of $\sim 50,000$ entries quite efficiently.



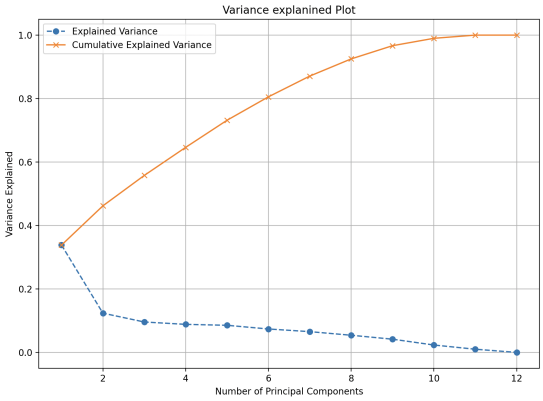
(a) t-SNE 2D reduction

(b) PCA 2D reduction

Figure 1: t-SNE plot on the left, PCA plot on the right

Looking at the shape of our data from the PCA output, it is hard to identify the main clusters. However, due to domain knowledge, we know there are 10 song genres to be considered: 'Electronic', 'Anime', 'Jazz', 'Alternative', 'Country', 'Rap', 'Blues', 'Rock', 'Classical', and 'Hip-Hop'. Thus, we can expect 10 clusters. It is set as a hyperparameter. Additionally, we can use k-means, because we do not

(a) K-means clustering on PCA 2D



(b) Variance Explained Plot

Figure 2: t-SNE plot on the left, PCA plot on the right

expect the clusters to be of obscure shapes, and thus we do not need to go with a density based approach like DBScan or other methods.

We can see how the clusters are grouped, as well as k means assigning outliers to the closest centroid. We performed a PCA reduction into 2 dimensions, but we can further compute the optimal PCA reduction using the explained variance. We can look to see where there is a fall-off in variance explained, and aim to account for 90% of the variance. We can see that at 9 principal components we account for over 90% of the variance, so we can use 9 principal components for our model.

Lastly, we can proceed with the classification. Using a XGBoost classifier, we can perform classification using all numerical predictors *popularity*, *acousticness*, *danceability*, *duration_ms*, *energy*, *instrumentalness*, *liveness*, *loudness*, *speechiness*, *tempo*, and *valence* to predict *music_genre*. When building this model, it is especially important we do not overfit, as songs are very different. XGBoost works well because it prevents over-fitting through regularization and iterative ensembles. Using the specified training and testing set with the PCA reduction, we construct a XGBoost model with multi-class decisions and then compute the AUROC along with the ROC curve for visualization. Additionally, I adjusted the learning rate hyper-parameter (0.1, 0.01, and 0.001) and found the best performance with learning rate of 0.1. Considering a relatively large dataset of 50,000 entries, I felt XGBoost was a suitable classifier maximizing efficiency due to the scalability of XGBoost as well as the accuracy with gradient boosting. The ROC curve tells us that our model is superior to random guessing (AUROC = 0.5), and has strong

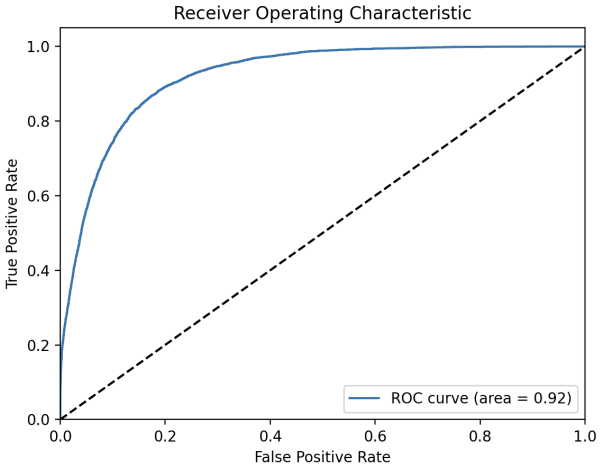


Figure 3: Plot for ROC curve

performance of an AUROC of 0.9208755066666667. The most important factor that seems to underlie

the model's success would be the selection of the number of principal components. For example, using 2 principal components although being the 2 largest eigenvalues results in a AUROC of 0.80. We can see from the figure of variance explained that 2 principal components account for $< 40\%$ of the variance, implying loss of data in the reduction. Additionally, we would make our model suceptible to underfitting since the data it is trained on does not capture the entire dataset.