Simone Rittenhouse
Prof. Pascal Wallisch
Intro to Machine Learning
17 May 2022

Capstone Project: Spotify Classifier

Highest AUC (Random Forest) = 0.8445

To classify Spotify songs into various genres, I built two models and compared performance between them. The two models were a random forest model and a gradient boosting model (i.e. AdaBoost). I found the random forest model performed best with an AUC of 0.84.

Pre-Processing

Before building either model, there were several pre-processing steps that needed to be addressed. Firstly, the random seed was set as my N-number: 17274546. Next, there were five rows in the dataset that were missing values across all columns. These five rows were dropped, leaving a total of 50,000 rows in the dataset. After dropping these null values, the target feature - 'music_genre' - was re-encoded using scikit-learn's LabelEncoder to make each genre a numeric label instead of a string. The columns 'artist_name' and 'track_name' were then dropped, as these linguistic variables were difficult to incorporate into a numeric model. Lastly, the column 'tempo' was found to be of type 'object' instead of being numeric. Upon closer examination, 4,980 rows of this feature were missing values and contained instead the character '?'. These missing values were dealt with after the train/test split by median imputation using the median tempo in the training and test sets respectively. Median imputation was chosen over mean imputation as the median is more robust to outliers.

In order to separate the data into a training and test set, the procedure detailed on the Spec Sheet was followed. 500 random indices were sampled without replacement from each genre of the dataset. These indices were then selected from the dataset to form a test set, while all indices not in the test set (i.e. the remaining indices) formed the training data. The re-encoded target labels, 'music_genre' were removed from both datasets to avoid using them as predictive features. After splitting the data, missing values of the 'tempo' feature were imputed as described above and the column was converted to type float. Additionally, the categorical variables 'mode', 'key', and 'obtained_date' were one-hot-encoded. However, 'obtained_date' had a one value in the training dataset that was not observed in the test set ('0/4') - leaving an unequal number of columns. This column was dropped from the training data to align the two datasets. Lastly, all features that were not binary variables were normalized by z-scoring in order to ensure the results of my dimensionality method, which was PCA, were legitimate and not skewed towards the mean or affected by unequal variances. It was also important to normalize the variables to handle the fact that the acoustic features were unlikely to be normally distributed.

Dimensionality Reduction

Some of the features in this dataset showed high correlations with one another, suggesting that not all of the information in the dataset was independent and that dimensionality reduction was advisable. Specifically, the highest positive correlation was observed between the variables ‘energy’ and ‘loudness’ (r = 0.84) while the highest negative correlation was between ‘energy’ and ‘acousticness’ (r = -0.79). In order to reduce the amount of dependent information in the dataset, PCA was conducted.

However, because PCA requires variables that can be meaningfully interpreted on a coordinate plane - there were some additional pre-processing steps needed to handle categorical dummy variables before using the algorithm. Namely, the algorithm detailed by Blaufuks (2022) was used to implement FAMD, which is more appropriate for mixed datasets containing both numeric and categorical variables. This algorithm was as follows: “a) Standard scale the numerical variables…b) For the categorical variables: Get the one-hot encoded columns, Divide each column by the square root of its probability sqrt(μ), Center the columns.” The probability of each column was computed by dividing the number of 1s observed in each one-hot encoded column by the length of the column, and columns were centered by subtracting the mean. This was performed separately for both the training and test data.

After this pre-processing, scikit-learn’s PCA class was fit to the training data, and the fit PCA was then used to transform both the training and test data. The test data was not used to fit a PCA of its own as it was important that both datasets were transformed the same way so that the principal components of both could be interpreted similarly.

To see how many factors were interpretable (and therefore useful for the model), the Kaiser criterion was used. As shown in **Figure 1**, 12 eigenvalues were found to be above 1. Therefore, the first 12 columns of the PCA-transformed training and test sets were used as the new training and test data. Additionally, the Loadings Plot in **Figure 1** shows that ‘energy’ and ‘loudness’ were the two features that were the most negatively-correlated with the first principal component (i.e. the component that explains the most variance in the original data), while ‘acousticness’ was the most positively-correlated feature. For the second component, ‘tempo’ and ‘danceability’ were the most positively and negatively correlated respectively.

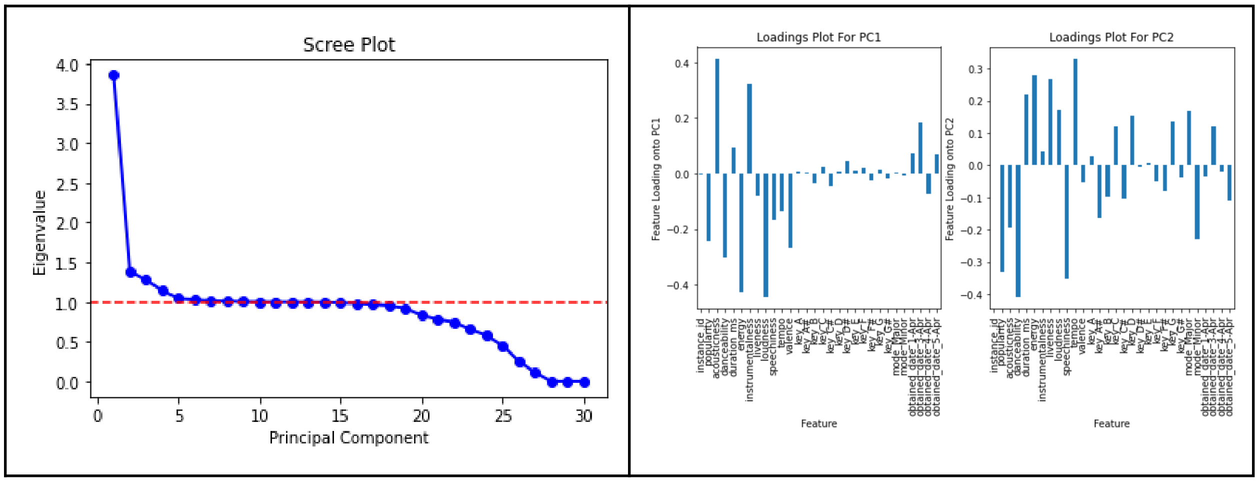


Figure 1. (Left) Scree plot showing the eigenvalues from PCA ordered by magnitude. (Right) Loadings plots for the first two principal components showing which features contribute most to each component.

Clustering

After reducing the dimensions of the training and test datasets down to shapes (45,000, 12) and (5,000, 12) respectively, k-means was used to cluster the data. This was chosen over density-based clustering because, after inspecting a 2d scatter plot of the first two principal components, the data seemed very tightly clustered, so density-based clustering likely would have found a single cluster. In order to determine the optimal number of clusters for k-means, the silhouette method was used. Cluster numbers between 2 and 10 were tested, and the silhouette plots can be seen in the **Appendix** section. For each k value tested, k-means was used on the new training dataset - defined as the first 12 principal components found applying PCA to the training data. Two clusters had the highest average silhouette score (0.2537) followed by ten (0.2094), nine (0.1973), and eight (0.1800) clusters. The optimal number was chosen as eight clusters as the silhouette plots of 2, 10, and 9 had clusters that were either extremely small, large, or barely at the average silhouette score - implying the clustering is not ideal.

K-means clustering was then fit to the PCA-reduced training data using k=8 with 100 randomly generated initial centroids and a maximum of 10,000 iterations for each run of the algorithm. These hyper parameters were set to be much higher than the default scikit-learn values because, again, the plotted data appeared to be amalgamated, so it seemed reasonable to account for this by allotting extra time for the algorithm to converge. After being fit to the training data, cluster labels were predicted for both the training and test data. These cluster labels were z-scored to prevent scaling issues and then added into both datasets as a feature so that the clustering could be used to help improve the performance of the classifiers.

Figure 2 shows the training data plotted along the first two principal components. On the left, the data has been labeled using the training target labels (i.e. the genres), and on the right, the labels have been assigned through k-means clustering. We can observe that there is some overlap between the clusters generated through k-means and the target labels. For example, Classical music loosely corresponds with cluster 2, and Rock music corresponds with cluster 7. Although in both cases clustering has been assigned in more than two dimensions, so the two-dimensional plots also show that many clusters/classes are superimposed. It's likely that more distinct clusters would be observable if this data was visualized in a higher dimensional space.

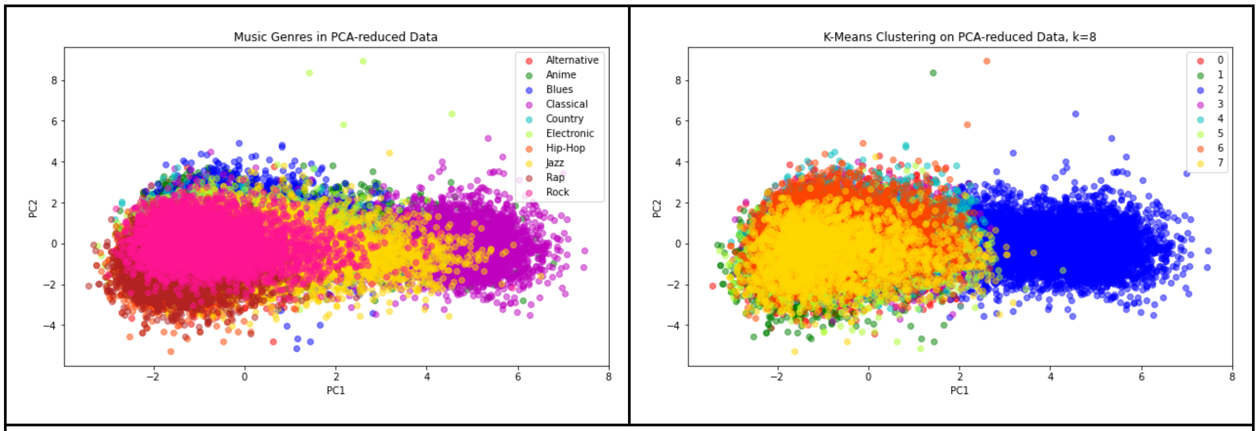


Figure 2. (Left) Training data plotted along the first two principal components and labeled by genre. (Right) Training data plotted along the first two principal components and labeled by k-means-generated cluster (k=8).

Classification

After adding the cluster labels to both the training and test data, a random forest and AdaBoost model were trained on the training data and their performance using the test data was compared to determine the best classifier between them. For both of these models, hyper parameter tuning was first conducted using the training data and scikit-learn's GridSeachCV class for 5-fold cross-validation. To account for these models being multi-class classifiers (rather than binary), the cross-validation optimized model parameters using 'f1_macro' - which is defined for a multiclass problem.

The optimal parameters from this were then used to build the models. For the random forest model, the optimal parameters were found to be 0.25, 0.5, and 100 for the maximum features to be considered at each split, the percentage of the data to use per sample (since bootstrap was set to True), and the number of trees respectively. The criterion to determine leaf purity and where to split was the Gini Index, since entropy requires the calculation of a logarithmic function and may have a longer runtime because of this. After using the training data and training labels to train the model, the model's performance was determined using the test data. Accuracy was found using the model's predicted class labels, and AUC was found using 'ovr' or one-vs-rest. This computes AUC for each class against all other classes, which is analogous to finding the average over each class' AUC where individual class AUC is determined using 1 where the true labels are that class' label and 0 for all other class labels. **Figure 3** shows the AUC across all classes. Classical music had the highest AUC of 0.96 and Alternative had the lowest AUC of 0.731. The average AUC was 0.844536, and the accuracy of the model was 0.3946.

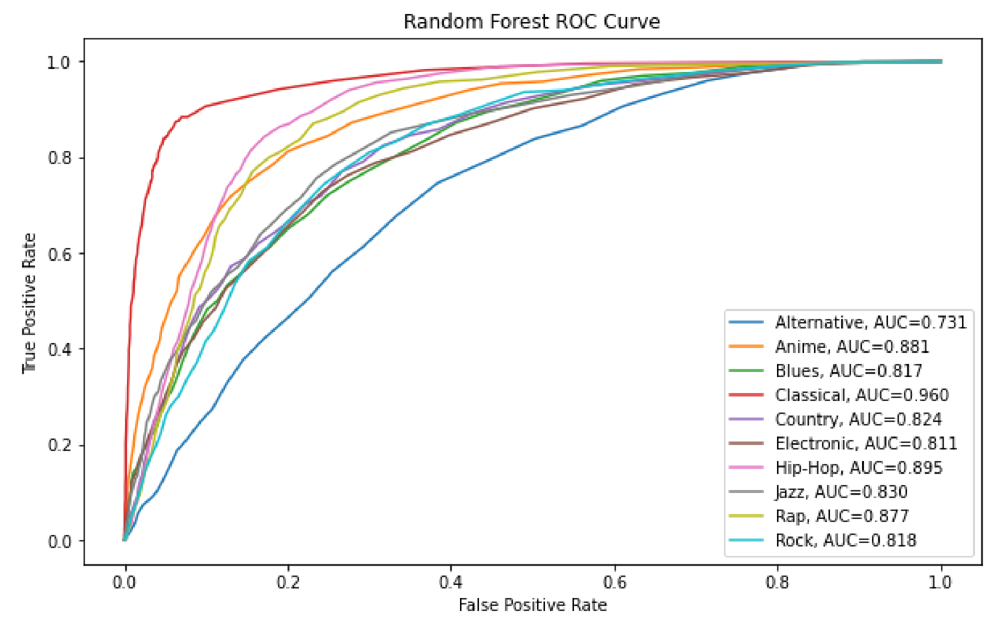


Figure 3. ROC Curve for each class using a Random Forest model

For the gradient boosting AdaBoost model, the optimal parameters were found to be 0.1 for the learning rate and 500 estimators, or stumps. Similar to the random forest model, these hyper parameters were used to train the model using the training data. Performance was then tested using the test data, and the AUC was again found using one-vs-rest. **Figure 4** shows the ROC curves for each individual class. The highest AUC was again found for Classical music (0.957), while the lowest was still Alternative music (0.535). The average AUC across all classes was 0.736440 while the accuracy of this model was 0.3368. For both metrics, AdaBoost performed slightly worse than the random forest model, meaning that the first model is the ideal classifier for this project.

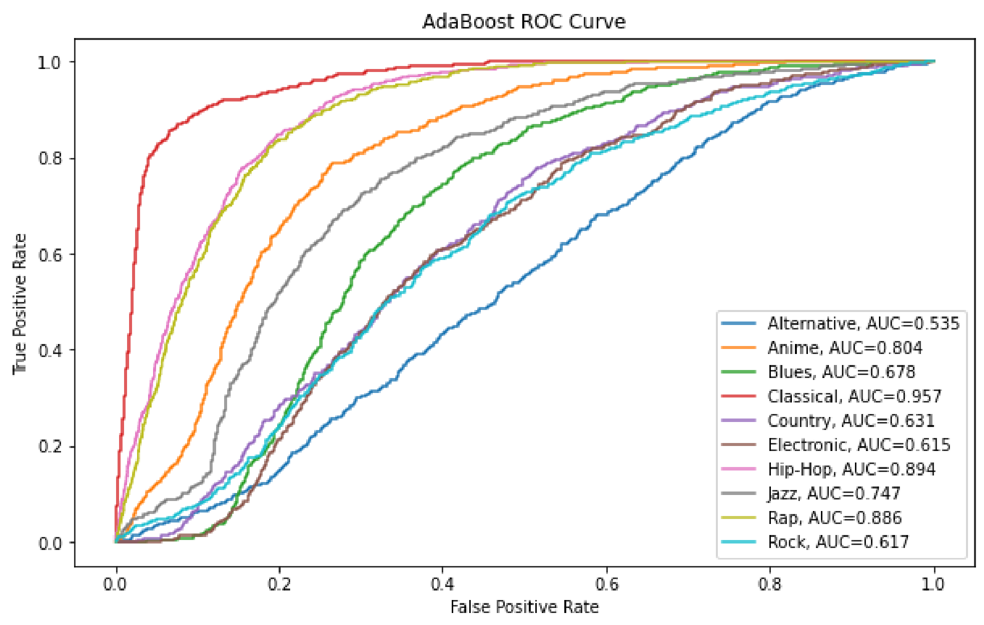


Figure 4. ROC Curve for each class using an AdaBoost model

Additional Analysis

To further explore the dataset, an additional analysis was conducted to determine if popularity differed significantly between genres or if the genres had around the same average popularity. To test this, the average popularity for each genre was found in the original, unprocessed dataset. These values can be seen in **Figure 5**, which shows that Rap is the most popular genre on average while Anime is the least popular. The most popular song was found to be "Wow." by Post Malone, and the least popular song was "Grohg: I. Introduction & Cortège - Entrance of Grohg" by Aaron Copland. To see if popularity was significantly different between genres, popularity for the most and least popular genres (i.e Rap and Anime) were tested using an independent samples t-test. The t-statistic of this was found to be 202.37, while the p-value was < 0.0001. This means that there was a significant difference in average popularity between genres, which can also be seen in **Figure 5** as Rap's average popularity is over twice that of Anime's.

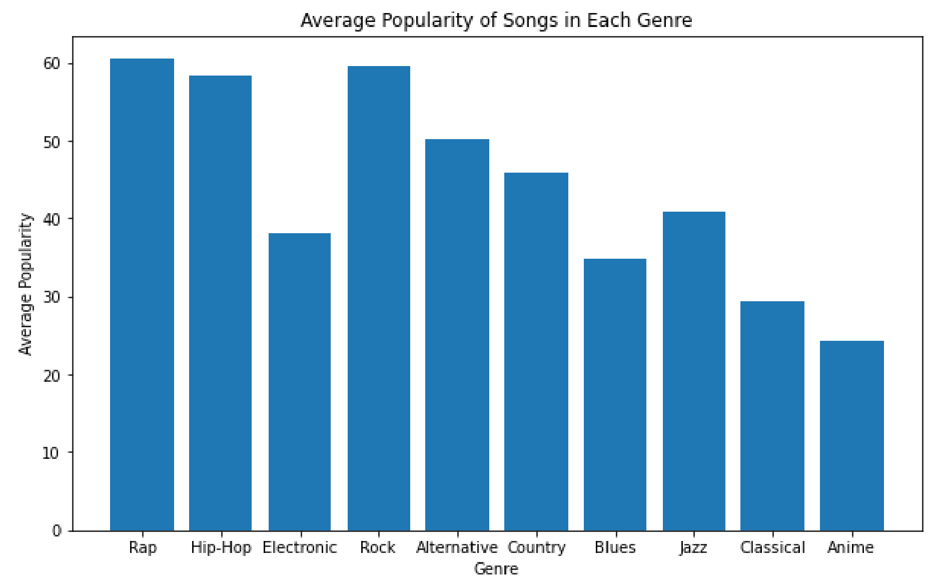


Figure 5. Average Popularity of Songs across Genres

Conclusion and Limitations

In summary, I built two models to classify Spotify songs into genres, and found that the random forest model outperformed the AdaBoost model both in terms of average AUC and accuracy. The AUC of the random forest model was 0.84 while the accuracy was 0.39. For both models, Classical was the highest performing class in terms of AUC while Alternative music was the lowest. Despite having a relatively high AUC, both models had low accuracy. This may be because, while the classifier had good performance at detecting the positive class, this was at the expense of performance on the negative class. A high false negative rate could therefore explain the poor accuracy of the models.

Aside from the low accuracy of the models, one other limitation to consider was that my results were variable based on the random seed used. For the silhouette plots in particular, I initially found that four clusters was an optimal number using a random state of 0 (before changing the seed to my N-number). This variability therefore had a significant impact on the result I obtained, and it should be noted that my results are conditional on the seed used and may not be robust. Lastly, hyper parameter tuning was conducted using the training data instead of a separate validation set in order to preserve the train/test split detailed in the spec sheet. However, because the models' hyper parameters were optimized on the training data, it's possible that these models overfit to the training data, which could have lowered performance on the test data.

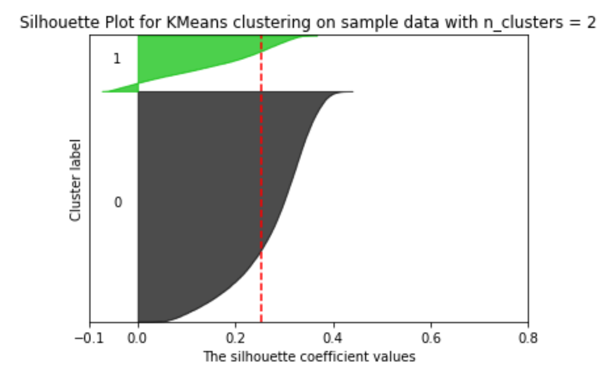
Overall, I found that the single most important factor that underlies the success of my models were the pre-processing steps taken - including dimensionality reduction and clustering. This was extremely important as dimensionality reduction helped remove dependent information, meaning that only features bringing useful, independent information were considered by the model. Additionally, clustering helped to provide a feature that, as shown by

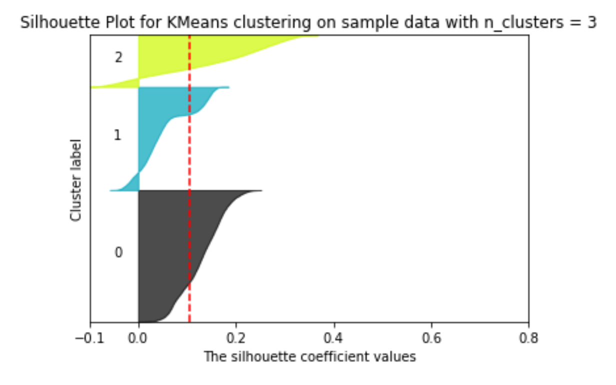
Figure 2, did have some overlap with the target labels, which also would have helped increase the performance of the model.

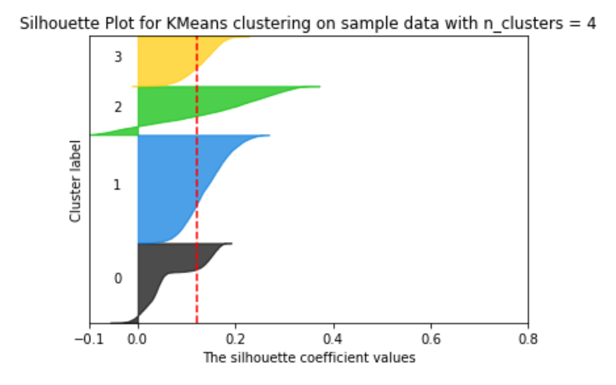
References

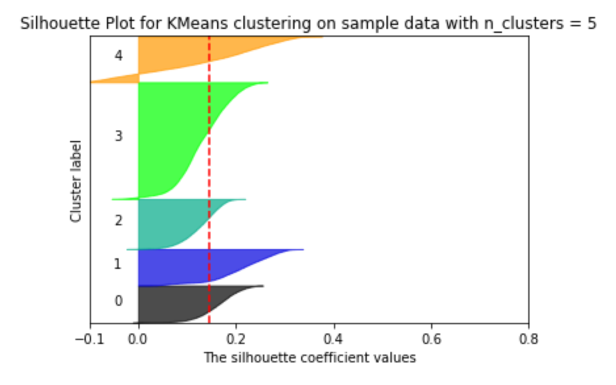
Blaufuks, W. (2022, March 28). *FAMD: How to generalize PCA to categorical and Numerical Data*. Medium. Retrieved May 12, 2022, from https://towardsdatascience.com/famd-how-to-generalize-pca-to-categorical-and-numerical-data-2ddbeb2b9210

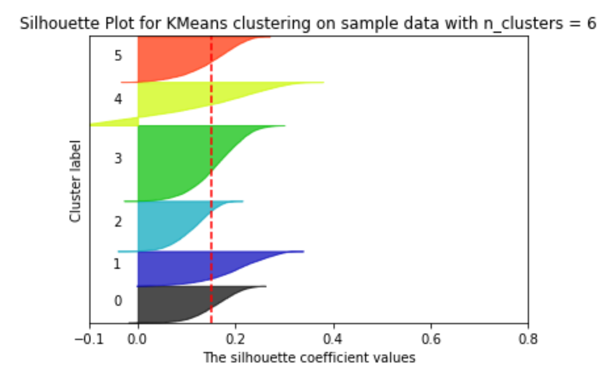
Appendix

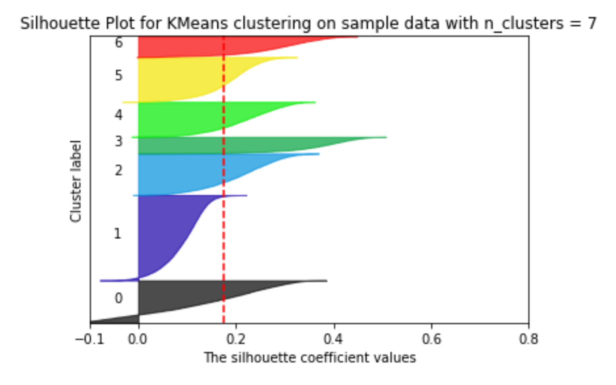

Silhouette Plot for KMeans clustering on sample data with n_clusters = 2
Cluster label
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values

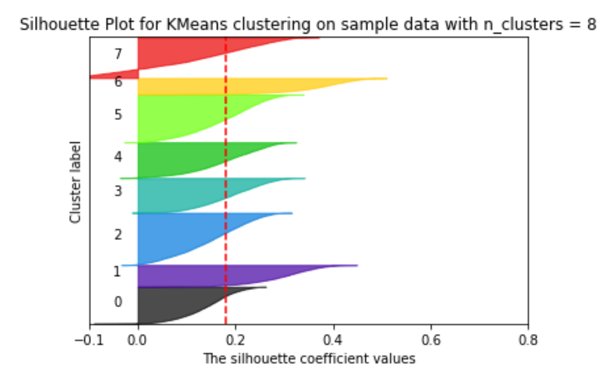

Silhouette Plot for KMeans clustering on sample data with n_clusters = 3
Cluster label
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values

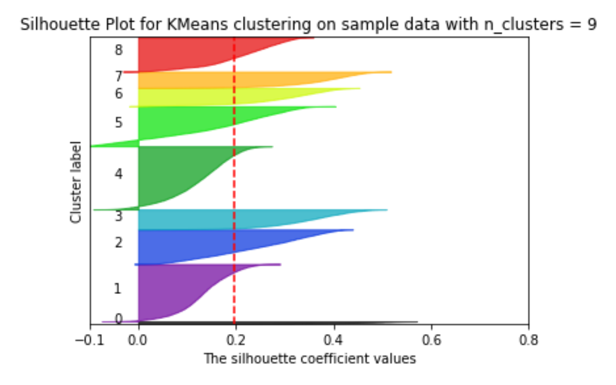

Silhouette Plot for KMeans clustering on sample data with n_clusters = 4
Cluster label
3
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values

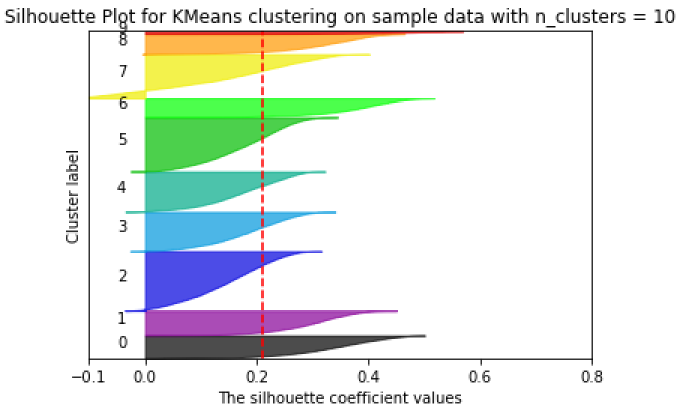

Silhouette Plot for KMeans clustering on sample data with n_clusters = 5
Cluster label
4
3
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values


Silhouette Plot for KMeans clustering on sample data with n_clusters = 6
Cluster label
5
4
3
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values


Silhouette Plot for KMeans clustering on sample data with n_clusters = 7
Cluster label
6
5
4
3
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values


Silhouette Plot for KMeans clustering on sample data with n_clusters = 8
Cluster label
7
6
5
4
3
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values


Silhouette Plot for KMeans clustering on sample data with n_clusters = 9
Cluster label
8
7
6
5
4
3
2
1
0
−0.1 0.0 0.2 0.4 0.6 0.8
The silhouette coefficient values


Silhouette Plot for KMeans clustering on sample data with n_clusters = 10
Cluster label
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
The silhouette coefficient values