ML Pipelines Report

Nikhil Shanbhag

April 2025

1 Supervised Pipeline

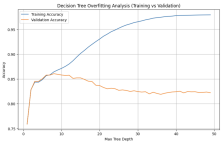
1.1 Data Preprocessing

The preprocessing pipeline began by dropping missing values. `Sex` and `income` were label encoded—`sex` being binary and `income` an ordinal target. Nominal categorical features like `workclass`, `marital-status`, `occupation`, `relationship`, `race`, and `native-country` were one-hot encoded. `Education` was dropped in favor of the numeric `education-num`, and `fnlwgt` was removed as it added noise rather than predictive value. Finally, the `native-country_Holand_Netherlands` column was dropped from the training set to match the test set.

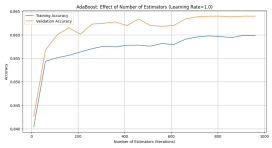
1.2 (a) Tree and Tree-Based Ensembles

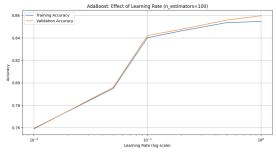
1.2.1 Part i: Hyperparameters

For decision trees, varying the tree depth made overfitting very easy, and by the time the depth got to 30, there was significant overfitting as seen below.

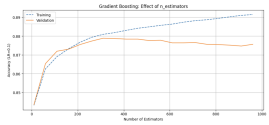


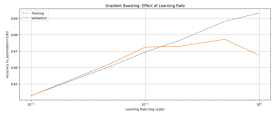
AdaBoost, on the other hand, produced the opposite trend – the validation accuracy was seemingly always above the training accuracy, making overfitting not possible. Here is what the visuals look like when varying the number of estimators and the learning rate:



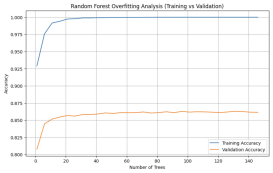


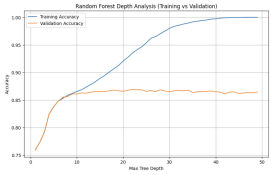
Gradient boosting began to overfit as the number of estimators and learning rate became very large, but not significantly. Thus, it seems that boosting methods in general are more difficult to overfit. Here is a comparison of the training and validation accuracy against the number of estimators and the learning rate:

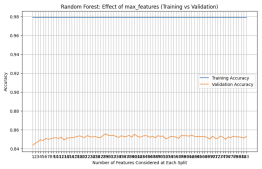




Finally, it seemed surprisingly simple to overfit random forests by varying the number of trees and the max depth of the tree. Also, when varying the number of features considered at each step without any max depth, the random forest is perfectly learning the training data, leading to an accuracy of nearly 100% on the training set, but much lower accuracy on the validation set. Here are those three plots:







1.2.2 Part ii: Accuracy and Runtime

For every method, the data was split into a training set and validation set, which was comprised of 20% of the data. After a validation method was used to determine the optimal hyperparameters, the performance on the test set was used to determine the best method.

Decision Trees The following hyperparameter grid was fit:

- **max_depth**: 3, 5, 7, 10, 15, 20, `None`
- **min_samples_split**: 2, 5, 10
- **min_samples_leaf**: 1, 2, 4
- **max_features**: `'sqrt'`, `'log2'`, `None`

Since decision trees are computationally efficient, it was possible to use a grid search with 5-fold CV, for a total of 945 fits.

AdaBoost The following hyperparameter grid was fit:

- **n_estimators**: 50, 100, 200

- **learning_rate**: 0.1, 0.5, 1.0
- **algorithm**: `'SAMME'`, `'SAMME.R'`

Since there are fewer parameters and therefore fewer fits to be done, this was computationally efficient. Thus, a 5-fold CV grid search was performed again, requiring only 90 fits.

Gradient Boosting The following hyperparameter grid was fit:

- **n_estimators**: 50, 100, 200
- **learning_rate**: 0.01, 0.1, 0.5
- **max_depth**: 3, 5, 7
- **min_samples_split**: 2, 5
- **min_samples_leaf**: 1, 2
- **max_features**: `'sqrt'`, `'log2'`, `None`

Now, since we are testing significantly more combinations and since gradient boosting requires a lot more time to tune than a decision tree, a 5-fold randomized search was done instead of a grid search. In order to improve accuracy, 50 iterations were performed, leading to a total of 250 fits.

Random Forests The following hyperparameter grid was fit:

- **n_estimators**: 50, 100, 200
- **max_depth**: 5, 10, 20, `None`
- **min_samples_split**: 2, 5, 10
- **min_samples_leaf**: 1, 2, 4
- **max_features**: `'sqrt'`, `'log2'`
- **bootstrap**: `True`, `False`

Again, randonm forests are computationally expensive for a large number of parameters, so a 5-fold randomized search was done with 50 iterations, leading to 250 fits.

Below is a final summary of the optimal parameters, test accuracy, and tuning time of each method based on the hyperparamaters and validation methods explained above.

Method	Best Parameters	Test Accuracy	Tuning Time
Decision Tree	max_depth: 10 max_features: None min_samples_leaf: 4 min_samples_split: 2	0.8577	1m 6s
AdaBoost	algorithm: SAMME learning_rate: 1.0 n_estimators: 200	0.8577	1m 52s
Gradient Boosting	n_estimators: 200 min_samples_split: 5 min_samples_leaf: 2 max_features: None max_depth: 5 learning_rate: 0.1	0.8724	12m 50s
Random Forest	n_estimators: 200 min_samples_split: 5 min_samples_leaf: 2 max_features: sqrt max_depth: None bootstrap: False	0.8648	7m 2s

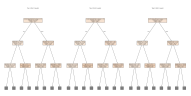
Based on this, we see that **gradient boosting** performs the best by a clear margin, with an accuracy just above 87%, followed by random forest with an accuracy of about 86.5%. This is likely because gradient boosting builds trees sequentially, each one correcting the errors of the previous, allowing it to model complex patterns effectively. While random forest also reduces variance through bagging, gradient boosting's iterative error correction gives it a slight edge in predictive performance. Decision trees and AdaBoost both achieved the same exact accuracy, but were clearly not as effective.

1.2.3 Part iii: Interpretation

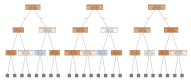
Decision Trees Here is how the classification tree and the important features barplot looks. The top 5 features in order of importance are martial status (married), number of years of education, capital gain, capital loss, and age.

AdaBoost Here is how the classification tree and the important features barplot looks. The top 5 features in order of importance are capital gain, martial status (married), number of years of education, age, and capital loss.

Gradient Boosting Here is how the classification tree and the important features barplot looks. The top 5 features in order of importance are martial status (married), capital gain, number of years of education, age, and capital loss.



Random Forest Here is how the classification tree and the important features barplot looks. The top 5 features in order of importance are capital gain, martial status (married), number of years of education, age, and number of hours worked per week.



All four methods identify similar features as important, with marital status (married), capital gain, and education consistently appearing in the top three. Age also appears frequently, suggesting that it has a strong predictive relationship with income. Although the exact order varies slightly and the random forest includes hours worked per week instead of capital loss, the overlap in key features between methods suggests that these variables play a central role in determining income classification. Also, it should be noted that the random forest classifier definitely distributed the importance of the characteristics more evenly than the other methods, as seen in the bar graph.

1.3 (b) Feedforward Neural Networks

1.3.1 Part i: Hyperparameters

In order to make the process computationally efficient, we can randomly vary each of the hyperparameters while keeping others constant. Therefore, I chose to do the following:

(1) Single hidden layer with ReLU activation, which allowed me to test a varying number of hidden units: 32, 64, 128. The learning rate was kept constant at 0.001 and the optimization method used was Adam.

(2) Two hidden layers with sigmoid activation, which allowed me to vary the number of hidden layers and the activation method simultaneously. I tested 64 and 128 hidden units, keeping the learning rate at 0.001 and the optimizer type as Adam.

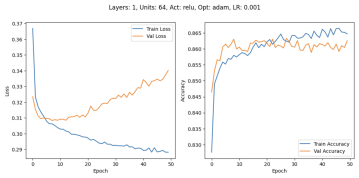
(3) Different activation functions while holding all else constant. This allowed me to test ReLU, sigmoid, and Leaky ReLU with 1 hidden layer, 128 hidden units, Adam optimizer, and learning rate 0.001.

(4) Different optimizers with varying learning rates. The optimizers tested were Adam and Stochastic Gradient Descent (SGD); the learning rates tested were 0.01, 0.001, and 0.0001. I used 1 hidden layer, 128 hidden units, and ReLU activation to test all of these.

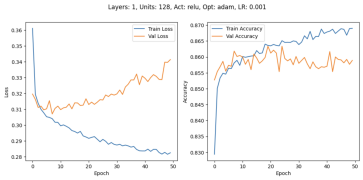
(5) Different random initializations. I used different initialized seeds randomly generated by PyTorch in order to see if there are any differences in the validation error. To test this, I used 1 hidden layer, 128 hidden units, ReLU activation, Adam optimizer, and learning rate 0.001.

Based on this randomized grouping and selection of hyperparameters, the two best combinations turned out to be (a) 1 hidden layer, 64 hidden units, ReLU activation, Adam optimizer, learning rate 0.001 and (b) 1 hidden layer, 128 hidden units, ReLU activation, SGD optimizer, 0.001 learning rate.

The combination in (a) led to a training loss of 0.2882, validation loss of 0.3402, training accuracy of 0.8647, and validation accuracy of 0.8626. Here is the visualization:



The combination in (b) led to a training loss of 0.3042, validation loss of 0.3103, training accuracy of 0.8579, and validation accuracy of 0.8620. Here is the visualization:



1.3.2 Part ii: Accuracy and Runtime

In order to train an MLP, I used the scikit-learn library instead of PyTorch. These can be computationally expensive, so I decided to use a subset of reasonable hyperparameters for the grid as follows:

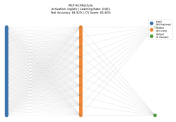
- **Architecture**: (64,), (128,), (64, 32), (128, 64), (256,), (128, 128)
- **Activation Functions**: relu, tanh, logistic

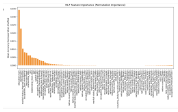
- **Optimizers**: adam, sgd
- **Learning Rates**: 0.001, 0.01, 0.1
- **Momentum (for SGD)**: 0.8, 0.9
- **Regularization (Alpha)**: 0.0001, 0.001, 0.01
- **Batch Size**: 32, 64, 128
- **Random State**: 42, 123, 321

To balance computation with accuracy, a randomized grid search was conducted with 5-fold CV and 50 iterations, similar to the tree and tree-ensemble methods. Additionally, the MLP classifier was built with an early stopping mechanism and a maximum of 500 iterations. Here are the optimal hyperparameters generated by this process:

- 1 hidden layer
- 64 hidden units
- Sigmoid (logistic) activation function
- Adam optimizer
- Learning rate 0.001
- Random state 123

The test accuracy turned out to be **0.8492**, which is slightly lower than the tree and tree-ensemble methods from above. Additionally, the time to train and tune the MLP was **31m 28s**, which is significantly longer than the tree-based methods. The visualization of the MLP and the important features selected are shown below.





It appears that the most important predictors (in order) based on the mean accuracy decrease when shuffled are years of education, capital gain, marital status (married), having an occupation of executive managerial, and number of hours per week worked. This is is different from what the tree-based and boosting methods showed, but the top three predictors remain the same.
Given the trade-off between accuracy and runtime, the tree-based ensemble methods appear to be more effective for this dataset, offering both better performance and reduced computational cost. This is likely because tree-based ensemble methods are better for these non-complex datasets with tabular data.

Furthermore, the fact that the MLP needed only 1 hidden layer, 64 hidden units, and chose the sigmoid activation function, which is generally less stable than ReLU, indicates that neural networks are probably unnecessary for this dataset.

With more computational resources, further improvements could come from using batch normalization, dropout, or learning rate schedulers for better training stability. Trying optimizers like RMSprop or AdamW may also help. Finally, ensembling MLPs or enhancing feature engineering could boost accuracy by capturing more complex patterns.

2 Unsupervised Pipeline

2.1 (a) Data and Pre-processing

Dataset I chose a dataset from Kaggle: https://www.kaggle.com/datasets/gauthamvijayaraj/spotify-tracks-dataset-updated-every-week.

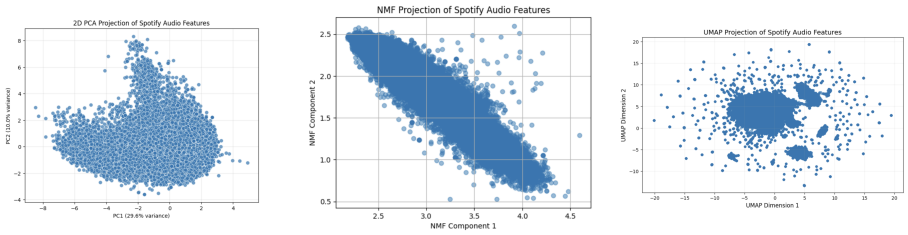
This datast gives Spotify tracks in six different languages, with over 60,000 rows and 20 features. Each track, which represents a row of data, has audio features associated with it.

Objective The objective of my machine learning model is to cluster tracks based on their acoustic features, which include the following: acousticness, danceability, duration, energy, instrumentalness, key, liveness, loudness, speechiness, tempo, time signature, and valence. Thereafter, we can understand how to group the songs into categories based on these features, which will allow us to create recommendations for playlists. As such, this is an unsupervised ML task.

Wrangling and Pre-processing The pre-processing pipeline first involved dropping uneccessary columns so that the data is filtered down the to the features that can be used for clustering. After, that missing values were dropped. As I was conducting my analysis, I noticed that there were loudness values in the dataset that were not possible, well below the -60 dB threshold. They caused erratic clusters, showing that some of the data was corrupted. Therefore, I decided to filter out these impossible loudness values outside the -60 dB to 0 dB range. After that, categorical features were encoded appropriately and only the numeric features were included for analysis.

2.2 (b) Exploratory Data Analysis and Visualization

To determine the most effective way to reduce the data's dimensionality, visualizations using PCA, NMF, and UMAP were generated. Below are the resulting plots:



As seen, PCA compresses the variance into a tight cluster with limited separation, indicating that linear methods may not capture the complexity of the data well. In contrast, NMF revealed a strong negative correlation between components, which is not ideal, and UMAP uncovered more defined structures—including a central cluster and several smaller ones—suggesting potential groupings based on acoustic features. These observations support the use of nonlinear dimensionality reduction techniques and clustering algorithms that can detect complex, non-convex structures. Therefore, even though I ultimately decided to use PCA when testing out different models, it became apparent to me that UMAP reduction will likely perform better.

2.3 (c) Modeling and Model Validation

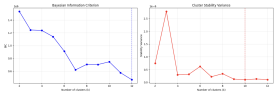
Based on EDA and the structure of the data revealed through UMAP and NMF, I chose to fit K-means clustering and Gaussian mixture models. These methods assume convex or elliptical clusters, which aligned well with the roughly globular shapes observed in the projections. I opted not to fit spectral or hierarchical clustering methods, as the dataset is extremely large, making these approaches computationally expensive (if not impossible) and less scalable, even with subsampling. Moreover, spectral clustering is more effective for data with complex graph-like structures, which were not strongly evident in the EDA. K-means and mixture models also allow for greater flexibility in tuning and evaluation.

For Gaussian Mixture Models (GMM), I chose to use full and tied covariance types because they allow for capturing more flexible and realistic cluster shapes in our high-dimensional data. The full covariance setting enables each component to have its own general covariance matrix, which is ideal when the data has elliptical clusters with potentially different orientations. Tied covariance, on the other hand, assumes all components share the same covariance matrix, providing a compromise between flexibility and computational efficiency. I avoided spherical and diagonal covariance types because they assume overly simplistic structures—either circular clusters (spherical) or axis-aligned ellipses (diagonal) — which did not align with the cluster shapes observed during EDA.

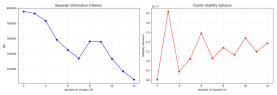
I used a BIC curve in order to understand what value of k best minimizes the BIC value, and I also generated a cluster stability variance plot in order to find the value of k such that the variance in stability is minimized. I tested

all values of $k \in [2, 12]$, seeing as there are 12 features we are interested in analyzing. It is likely that the plots will choose higher values of k as optimal for both plots. However, as $k \to 12$, the model will likely become less interpretable, and as such, the Silhouette score will decrease due to significant cluster overlap. Therefore, I tried to strike a balance between the model minimizing BIC score, maximizing stability, and being as interpretable as possible. Here are the plots for this specific method:

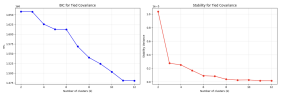
GMM with Full Covariance and PCA Reduction Based on the plots below, I chose $k = 7$ to strike the balance I described above. Clearly, there is a local minimum in the BIC plot and cluster stability plot at this value, and the values do not decline significantly thereafter. However, this turned out to be the weakest model out of all the ones I tested, which makes sense because PCA did not seem to separate the clusters much, achieving a silhouette score of 0.33.



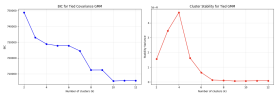
GMM with Full Covariance and UMAP Reduction For this one, I chose $k = 4$ because it is significantly more stable albeit I am trading off for a high BIC score. However, the clusters will be well-separated and interpretable. This happened to separate into clear clusters, but one of the clusters was comprised entirely of points that are outliers, as shown in the EDA. The silhouette turned out to be 0.39, which is reasonable for this dataset.



GMM with Tied Covariance and PCA Reduction Using a similar logic where I tried to balance BIC score, stability, and interpretability, I decided to use $k = 7$ clusters. Even though the silhouette score here is also 0.39, it was apparent that there was cluster overlap, which is expected for this dataset since varying types of songs likely have similar features. Nevertheless, this was an improvement over the full covariance GMM.

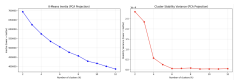


GMM with Tied Covariance and UMAP Reduction For this one, I chose $k = 8$. The silhouetter score was the highest among all the models I tested, at 0.44. Also, even though there was overlap, it was clear that there were distinctions between clusters, which allowed for interpretability.

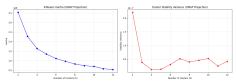


K-means clustering is very efficient computationally but might find it harder to cluster data points especially when the structure is non-linear. I used a similar logic to validate my K-means models, this time looking to minimize the inertia using an elbow plot, which is analogous to the BIC for GMMs. Additionally, a variance stability plot was also used in order to understand optimal values for k. Once again, both plots will favor higher values of k, but I looked to strike the balance between the elbow plot, stability plot, and interpretability. Here are the specific methods I used:

K-means with PCA Reduction The elbow plot seems to have a slight elbow at $k = 9$, which limits interpretability. Therefore, I decided to prioritize the stability plot, which shows that increasing after $k = 6$ does not change stability by much. Thus, I went with $k = 6$ clusters. Even though this utilizes entirely linear methods, it seemed to separate the clusters somewhat well and achieve a silhouette score of 0.36.



K-means with UMAP Reduction Once again, there is no clear elbow in the plot for a reasonable value of k. Therefore, I went with the stability plot, which suggests $k = 4$. Although combining K-means with UMAP may seem unconventional—since K-means performs best in linear settings while UMAP is a non-linear dimensionality reduction technique—this approach surprisingly yielded the second-highest silhouette score of 0.42.



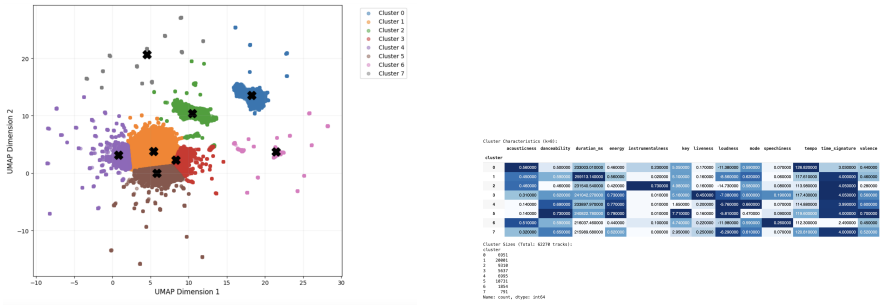
2.4 (d) Communication of Results and Interpretation

A silhouette score above 0.3 can be considered good for this dataset due to its complexity and the nature of the features. The dataset contains a real-world Spotify track data, which is inherently high-dimensional and noisy, making sharply separated clusters difficult to achieve. Additionally, many of the audio features—such as tempo, loudness, and energy—tend to overlap across genres and languages, which naturally limits the distinctness of clusters. Since this is an unsupervised learning task with no true labels, a silhouette score

above 0.3 indicates that the clustering method has uncovered some meaningful structure in the data.

Based on the visualizations from above, it appears that the best model is the **GMM with tied covariance, k=8, and UMAP Reduction**. Below are visual summaries for this model.

The first visual on the left is the song tracks broken into clusters and the second visual on the right shows how much each feature impacts each cluster, with higher values meaning more impact. The values in the matrix are computed by taking the mean of each audio feature for all tracks within a cluster.



The eight acoustic clusters reveal distinct musical profiles suited for different moods and settings. Clusters 0 and 2 (Introspective Acoustic and Instrumental/Classical) create calm, reflective atmospheres — ideal for relaxation, studying, or unwinding. Clusters 1 and 7 (Mainstream Pop and Upbeat Indie Rock) offer balanced, familiar energy for casual listening, commutes, or social gatherings. Clusters 4 and 5 (Dance-Pop and High-Energy Party) deliver vibrant, rhythmic tracks perfect for workouts, parties, or motivation boosts. Cluster 3 (Live Performance Energy) captures the excitement of concerts, great for immersive listening, while Cluster 6 (Spoken Word) provides engaging, talk-driven content for focused or transitional moments, like podcasts. Based on this, it makes sense why there is overlap between clusters 1, 3, 4, and 5.

Based on cluster size, we also gain an understanding of the type of music people prefer. The largest group, Cluster 1 (20,001 tracks), shows how people enjoy mainstream pop with balanced energy ideal for casual listening. Additionally, high-energy Cluster 5 (10,731 tracks) dominates workout and party playlists, very common all around the world.

These groupings allow for better playlist curation based on activity and mood rather than just genre or language. Whether seeking background music for work, high-energy tracks for exercise, or soothing sounds for relaxation, the clusters help make music playlists more personalized for individuals and larger groups of people.