Report for Final Project

Machine Learning

Team Members
Sayyor Yusupov
Arina Gepalova

Introduction

For this project, we decided to perform a classification of music genres using various features of a song. We are using Spotify Songs Dataset [1]. We have personal experience of listening to many songs of different genres and it was interesting to see if different genres have different characters that make them stand out.

Data Exploration

Preprocessing

Since the main dataset ("genres_v2.csv") contained 19 columns of mixed data types and a large number of rows (42.000), we explicitly set the data type for each column when loading it. This ensured correct data types.

Using describe() and unique() function, we found out that the columns "key", "mode", "time_signature" and "genre" are categorical variables. There are 12 unique values for "key", 4 unique values for "time_signature" and 2 unique values for "mode". After doing some research [4], we concluded that the column "key" refers to 12 musical notes (e.g. on a piano) and their initial ordering (integer values from 0 to 11) does not have any meaning. Similarly, "time signature" with unique values 1, 3, 4 and 5 refer to a rhythmic structure of a musical piece [5]. Therefore, for models that require numerical attributes (e.g. LDA), we use one-hot encoding to turn it into a set of numerical columns.

For our classification goal, the column "genre" is the target variable.

Regarding missing values, after plotting histograms for each numerical column (Fig. 1) and looking at the output of describe() function, we found out that there aren't any unreasonable maximum or minimum values. The only columns with empty values are the columns "song_name", "Unnamed: 0" and "title", with about the same number of missing values at around 50% of the whole dataset. We decided not to include these columns for our classification task due to the large proportion of

missing values and the columns giving away the genre of the song. Also, we cannot perform imputation of the textual columns “song_name” and “title”.

The number of missing values for “title” and “Unnamed: 0” are the same and the missing values are located at the same rows for both columns, which most likely means that they are related to each other. The column “title”, according to its values, seems to be the name of a playlist, if the song is in one. There is also an unnamed column that has an almost uniform distribution (Fig. 1).The unnamed column has all integers for non-missing values, so we believe it refers to the number of views of the given song in the given playlist. We renamed it to “view_count”.

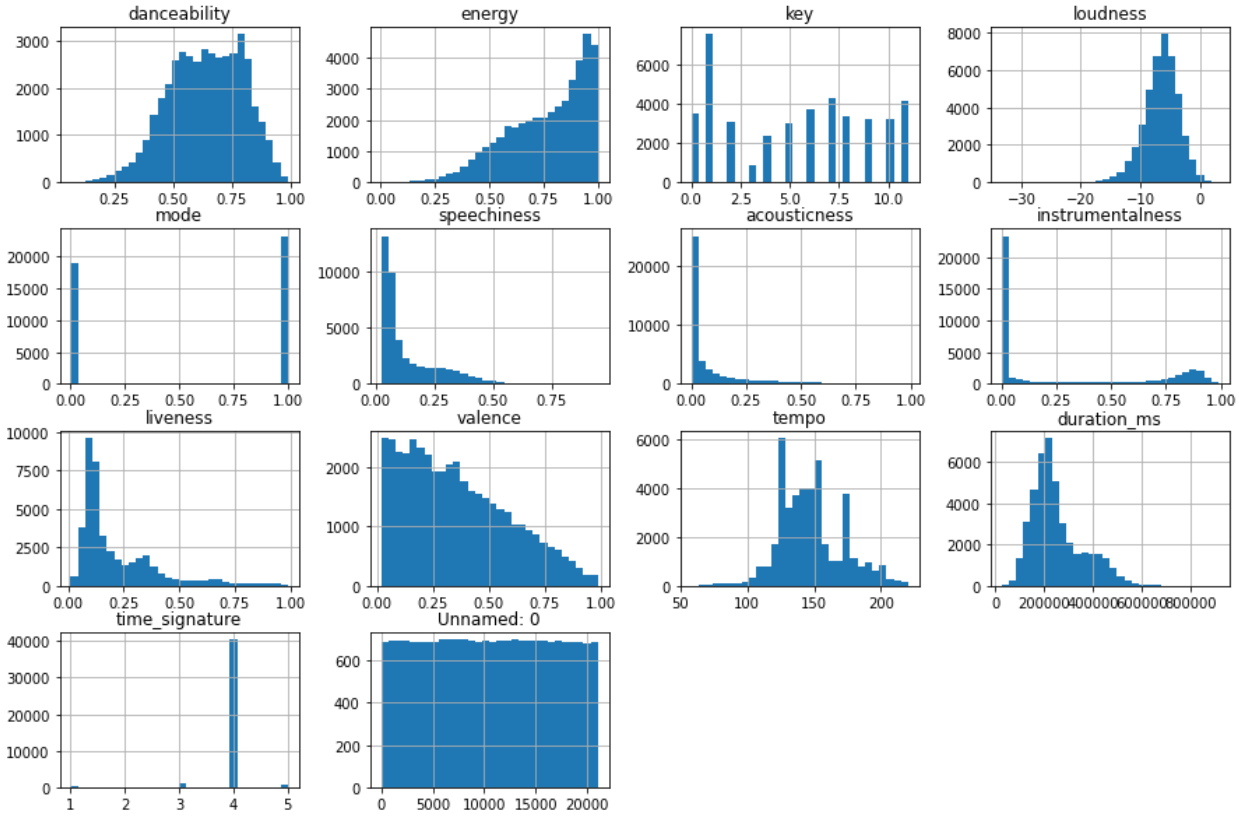

Figure 1. Histogram for all numerical columns of the dataset.

The columns “track_href”, “id”, “uri” and “analysis_url” are storing the same information, the url of the song track. We can further analyze this information to see if it can be useful, but we can keep only one of the columns. We also removed the column “type” as it contains only one unique value. Also, some of the ids of the “id” column get repeated. All of the columns have the same values except for “view_count” and “title”. Since earlier we decided not to keep these columns for classification, we kept only 1 instance of the song, having only unique values of “id”. As a result, we removed 15% of the repeated data.

After applying IQR, it seems there are quite a lot of outliers for columns “loudness”, “speechiness”, “acoustiness”, “liveness”, “duration_ms” and “time_signature”. Domain knowledge is required to assess whether they are outliers.

Feature selection / extraction

Misleading attributes can negatively affect the accuracy of a model.
First, we drew pairplots on the attributes (Fig 1, Fig 2) to use their relationships and how the data is separated into genres based on different pairs of attributes. For most of the attributes, the data points are mixed. However, the attributes “loudness” and “valence” seem to have some separation of the classes as can be seen from both their univariate distribution plots and scatter plots. The attributes “tempo”, “duration_ms”, have even more separation.

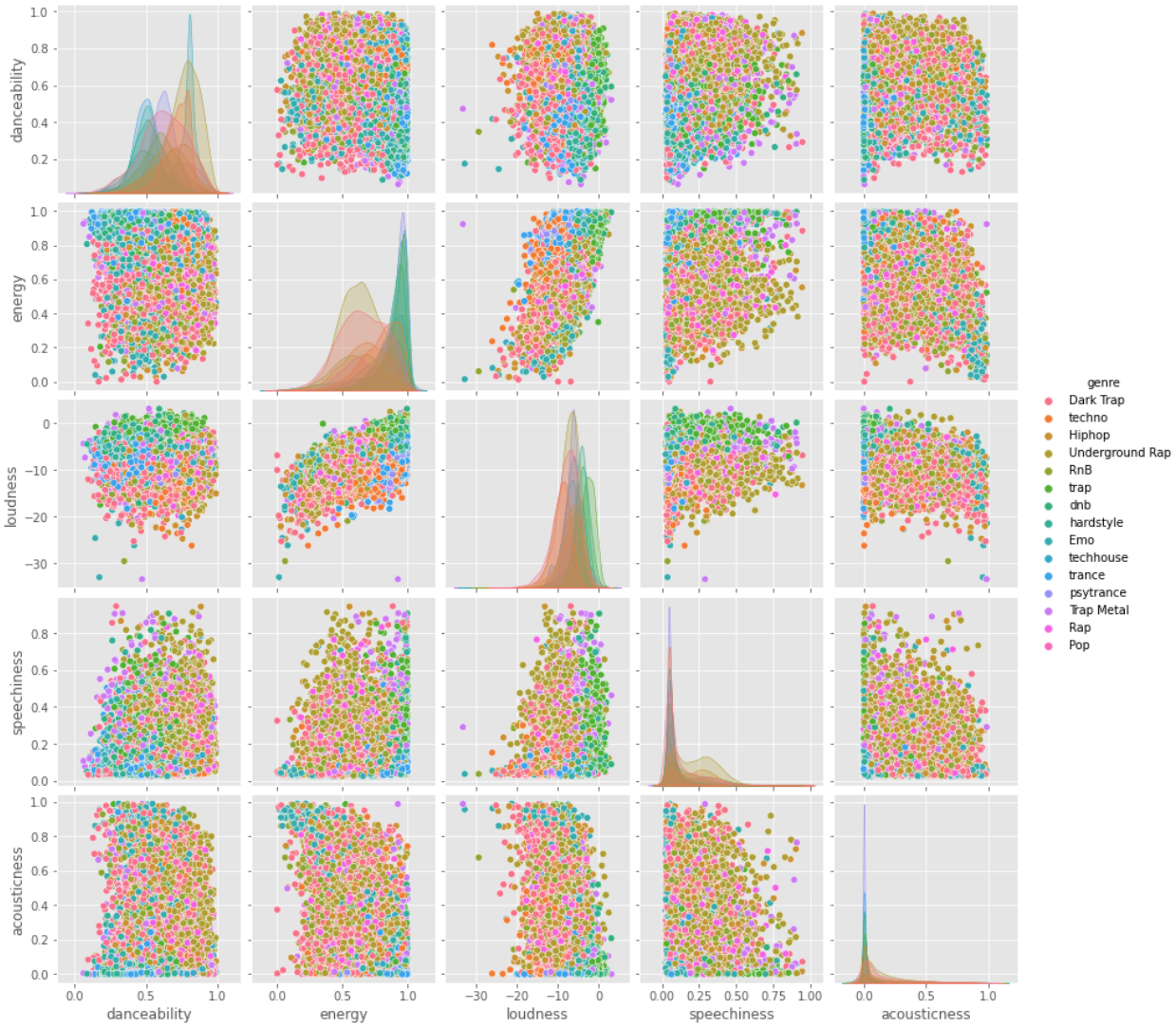

Figure 2. Pair Plot on the first 5 numerical attributes

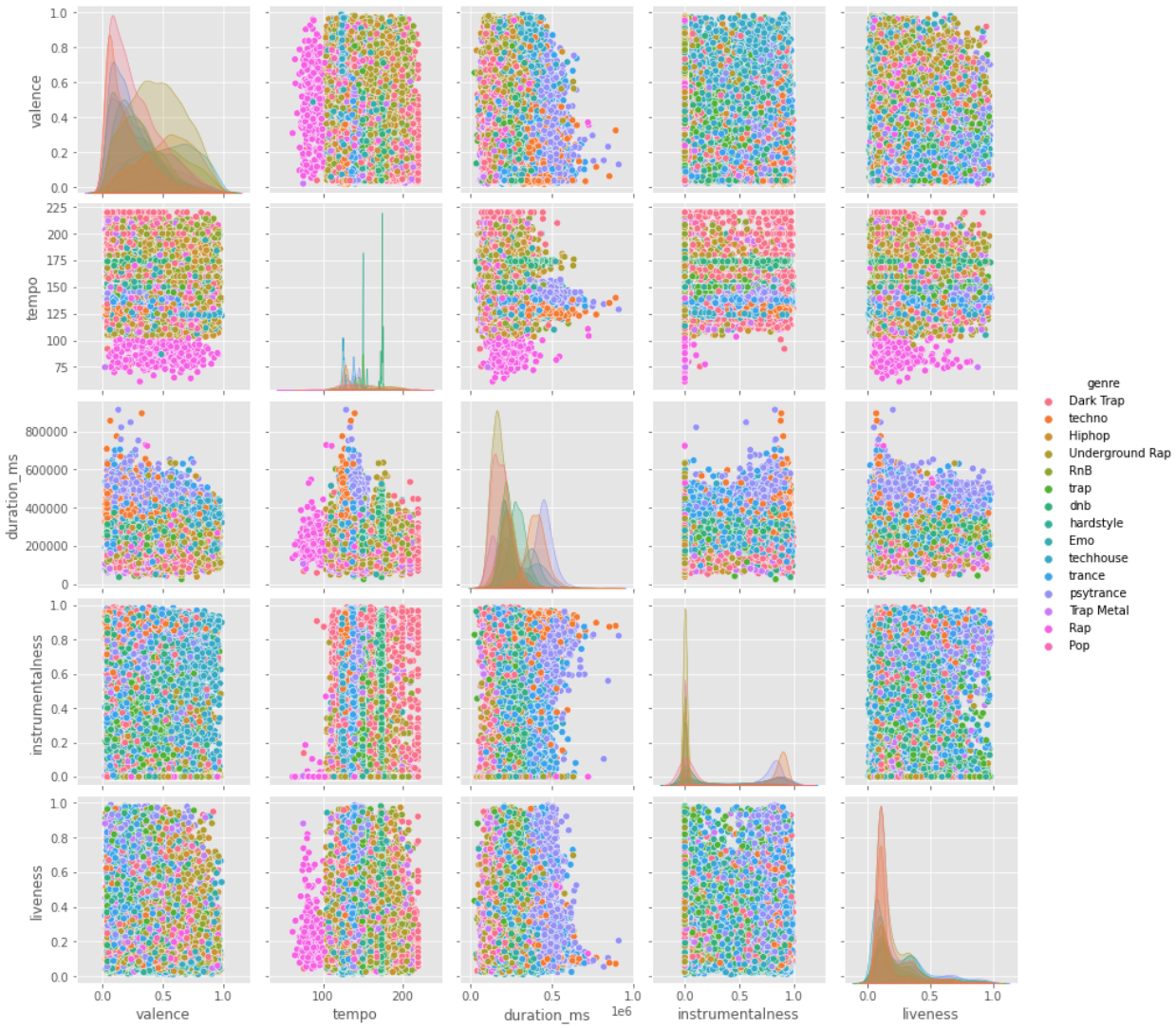

Figure 3. Pair Plot on the next 5 numerical attributes.

Next, we used Extra Trees Classifier, which is a variation of Random Forest, to approximate the Gini importance of each attribute in regards to the “genres” attribute. We tried to run the algorithm several times due to stochasticity of the algorithm but achieved the same results. According to the result (Fig. 4), top-5 most important features are “tempo”, “duration_ms”, “instrumentalness”, “danceability” and “energy”. This result confirmed our assumption about “tempo” and “duration_ms”. Also, we can see from distribution plots of the other 3 attributes (in Fig 1, Fig 2) that for some classes or some parts of classes, the separation of values is very clear which contributes to the attributes’ importance.

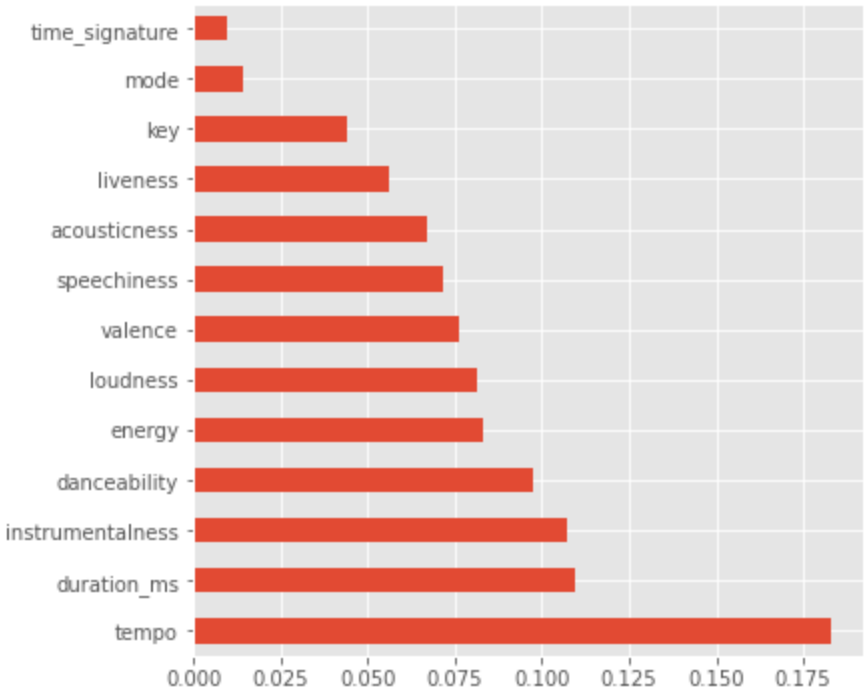

Figure 4. Feature Importance Parameter of Extra Trees Classifier.

Next, we tried to use Recursive Feature Elimination (RFE) which recursively applied the given model and removed the attributes that performed badly based on accuracy along the way. As the model to run we used Linear Discriminant Analysis, one of the algorithms we are considering for this project. RFE chose 6 attributes, namely “danceability”, “loudness”, “acousticness”, “tempo”, “duration_ms” and “time_signature”.

Modeling

We wanted to try out both simpler algorithms like K-Nearest Neighbors (kNN), and more advanced and high-performing algorithms like Random Forest Classifier. We also tried to run a simple Decision Tree to see how it compares to Random Forests. Furthermore, we chose to try algorithms that assume a linear relationship between input and output like Logistic Regression and Linear Discriminant Analysis (LDA) (and its counterpart with quadratic decision boundary Quadratic Discriminant Analysis (QDA)).

We apply standardization in the form of MinMaxScaler for Linear Models that we use, namely LDA, QDA, Logistic Regression, to make sure all attributes are equal during the learning process. We also apply MinMaxScaler to the Distance-based algorithm kNN. However, we do not apply standardization to Tree-based algorithms Random Forest and Decision Tree as they create conditions by splitting the values of attributes and do not depend on the magnitude of the attribute values. We also did not apply it to Naive Bayes as it assumes independence between features.

Validation Protocol

We are using Stratified K-fold cross validation to evaluate our models and tune the hyperparameters. This variation of K-fold cross validation maintains the same percentage of instances for each class. Initially, we set K to be 10 as it usually has low bias and a modest variance [2]. We also decided to make it Stratified to ensure that each fold contains the same number of values for each class.

We set the test data to be 20% of our dataset. To make sure there is no data leakage, we had to apply data preprocessing before each cross-validation iteration, i.e. fit for each new group of training folds and transform both training and validation folds. This can be achieved with sklearn.pipeline when using GridSearchCV [3].

Models Results

You can see the important metrics for predicting test data on different models in Figure 5. In particular, we optimized the number of important features used. The important features are selected with Extra Trees Classifier (they are labeled as Top-X in the table) and RFE, as mentioned above. RFE selects the features based on the specific model, while feature importance with Extra Trees Classifier is optimized for Extra Trees Classifier. However, we could not implement RFE for most of our models as they did not have a parameter to calculate the feature importance and a function for calculating should have been devised. Finally, we are also tuning the hyper parameters for each set of features chosen using GridSearchCV. Then we are predicting for test data and inputting the results to the table.

When tuning hyperparameters like learning rate and regularization parameters, we are using log space because it is more efficient at finding the optimal values than linear scale [7].

LDA

LDA is suitable for multi-class classification and, although assumes no correlation, can perform well even if correlation exists. For LDA, we tried to tune its hyperparameters “solver” and “shrinkage”. SVD was the best solver and regularization when tried with solver “lsqr” (as “svd” does not allow regularization) did not improve but worsened the accuracy. Therefore we set “shrinkage” to 0 and “solver” to “svd”.

QDA

QDA is a variation of LDA that uses curved shapes instead of lines and hyperplanes to separate classes. For this model, we are tuning only the regularization parameter that controls the degree of shrinkage of covariance matrices of individual classes. It can be a value between 0 and 1.

k-NN

For kNN, we are tuning the very important k-parameter, i.e. number of nearest neighbors to consider when considering a new instance, and metric, the distance measurement which is used to

determine the neighbors. We are choosing between “euclidean”, “minkowski” and “manhattan” distances.

Gaussian Naive Bayes

This algorithm requires numerical attributes. Because we have one-hot-encoding for our categorical variables, we can use all of our dataset. We are tuning its hyperparameter var_smoothing, “portion of the largest variance of all features that is added to variances for calculation stability” [8].

Logistic Regression

We are using an ElasticNet to see if the combination of Ridge and Lasso performs better, therefore we are also tuning the hyperparameter ‘l1_ratio’. We are tuning between the solvers ‘newton-cg’, ‘sag’, ‘saga’ as they are the only ones that can handle multiclass problems and ElasticNet. We are also tuning the regularization parameter C. Since the algorithm already performs feature selection and we are further tuning it (value C), we do not need to use the Feature Importance we did for other algorithms.

Decision Tree

For this model, we considered the hyperparameters criterion, to measure the quality of a split, in particular to check how much information was transferred after the split. We also tuned the maximum depth of the tree, minimum number of samples needed to split a node, minimum number of samples for a leaf node and the number of features to consider when looking for a split. To prevent overfitting, but still find the best hyperparameters, the values for “min_samples_split” and “min_samples_leaf” were set up to 5 and the value for “max_depth” up to 30.

Random Forests

In addition to the hyperparameters in Decision Tree, we also tuned the number of trees to generate (n_estimators) and weights assigned to classes (class_weight).
We only fine tuned parameters of Decision Trees and Random Forests once and did not fine them for feature selection because of the enormous amount of time (over an hour) it takes to tune the many hyperparameters. Also, Decision Trees prioritize important attributes for their construction so feature selection already exists in them. We did, however, tune feature selection with already selected hyperparameters. Although Decision Tree did not have better results, Random Forests performed better with fewer important features, as can be seen in the table (Figure 5).

Ensembles

We wanted to see if the accuracy could be improved if we get the prediction based on voting of multiple algorithms, Majority Voting Ensemble. This method is useful only when all the models are already performing relatively well and agree on most of the instances [9]. For the number of models we are using numbers: 3, 5, 7, since if it is even, there will be more cases of when the votes are equal for multiple classes. We chose number of important features as 10 because the best performing models on average use this number, according to our results.

Algorithm	Features used	Parameters chosen (with Grid Search)	Mean Accuracy (%	F1 Macro (%)	Precision Macro (%)	Recall Macro (%)
LDA	**All**	**Solver=svd, Shrinkage=0**	**53.5**	**48.9**	**51.7**	**49.1**
	Top-3	Solver=svd, Shrinkage=0	34.4	25.0	27.0	28.4
	RFE top-6	solver=lsqr, shrinkage=0.42	46.5	37.3	36.0	40.1
	Top-5	Solver=svd, Shrinkage=0	45.8	39.2	42.6	40.1
	Top-7	Solver=svd, Shrinkage=0	50.9	44.7	46.9	45.5
	Top-10	Solver=svd, Shrinkage=0	53.2	48.3	51.0	48.6
	Top-11	Solver=svd, Shrinkage=0	53.3	48.6	51.5	49.0
QDA	All	reg_param=9	61.7	57.8	58.6	59.1
	Top-3	reg_param=0	48.1	39.4	42.1	42.3
	Top-5	reg_param=0	57.4	51.6	55.0	52.7
	RFE top-6	reg_param=0	54.8	47.4	50.6	49.4
	Top-7	reg_param=0	60.9	55.9	64.2	56.9
	Top-8	reg_param=0	62.3	58.2	60.9	59.2
	Top-9	**reg_param=0**	**62.4**	**58.6**	**60.7**	**59.5**
	Top-10	reg_param=0	62.0	57.9	59.4	59.1
kNN	All	n_neighb=7 met.=manhattan	57.1	52.5	53.5	52.6
	Top-3	n_neighb=20 met.=manhattan	52.8	46.0	48.1	47.1
	Top-5	n_neighb=15 met.=manhattan	59.2	53.4	57.4	53.9
	Top-7	n_neighb=10 met.=manhattan	61.3	55.8	60.2	55.8
	Top-8	**n_neighb=7 met.=manhattan**	**61.4**	**56.9**	**58.3**	**56.9**
	Top-9	n_neighb=7 met.=manhattan	61.3	56.7	57.8	56.8

	Top-10	n_neighb=7 met.=manhattan	60.2	55.6	57.4	55.9
Gaussian Naive Bayes	**All**	**var_smoothing= 0.000433**	**49.4**	**44.6**	**48.3**	**48.4**
	Top-3	var_smoothing= 0.000002	44.3	41.3	49.4	45.0
	Top-6	var_smoothing= 0.0001	48.7	44.5	49.2	48.3
	Top-9	Var_smoothing=0.000007	49.6	44.5	48.0	48.5
	Top-11	var_smoothing= 0.000019	47.2	43.5	49.4	47.2
	Top-10	var_smoothing= 0.000004	45.2	42.0	49.2	45.7
Logistic Regression	**All**	**C=10, solver=saga, l1_ratio=1.0**	**58.9**	**53.4**	**56.8**	**53.4**
	Top-10	C=10, solver=saga, l1_ratio=0.8	58.1	52.1	53.4	52.3
	Top-8	C=10, solver=saga, l1_ratio=1.0	57.0	51.1	52.5	51.2
Decision Tree	**All**	**criterion=gini, max_depth=15, min_samples_split=2, min_samples_leaf=5, max_features=None**	**61.7**	**57.6**	**59.3**	**57.0**
Random Forest Classifier	All	n_estimators=500, max_depth=None, min_samples_split=5, min_samples_leaf=2, class_weight= balanced	67.9	65.3	64.5	66.5
	Top-10	**same as above**	**68.1**	**65.4**	**64.8**	**66.5**
	Top-11	Same as above	67.9	65.4	64.7	66.5
	Top-12	Same as above	67.8	65.2	64.7	66.3
	Top-9	Same as above	67.9	65.2	64.6	66.3
	Top-8	Same as above	67.1	64.1	63.2	65.3
Ensemble	**Top-10, all models**	**Best hyperparameters above, voting=hard**	**65.7**	**61.6**	**62.5**	**61.7**
	Top-10 feats, top-5 models	Same as above, voting=hai	62.2	57.7	58.7	58.1
	Top-10 feats,	Same as above + weights =	65.1	60.2	61.6	60.4

	All models	[68.1, 62.4, 61.7, 61.4, 58.9 53.5, 49.2] (models accuracies), voting=hard				
	Top-10 feats, All models	voting=soft	66.2	61.7	63.2	61.7

Figure 5. Results of evaluating the different models with different numbers of top features on the test dataset. The parameters are chosen with GridSearch.

Comparison of the Models' Results

To summarize, the best performing models (from best to worst) by mean accuracy are Random Forest Classifier (68.1), QDA (62.4), Decision Tree (61.7), kNN (61.4), Logistic Regression (58.9), LDA (53.5) and Gaussian Naive Bayes (49.4).
kNN, as expected, performed best when we selected only several most important features, in this case Top-8. When more less important features were added, it suffered from the curse of dimensionality and performed worse.
Also the model that assumes independence between features, Gaussian Naive Bayes, performed worse than others, while many of our features are closely related to each other.
QDA performed significantly better than LDA. Therefore, we can assume that the relationship between input and output is not linear and QDA having different covariance matrices for each class and having a quadratic decision boundary helped to separate classes better.

Next, we drew confusion matrices for each model to check the accuracy for each particular class (Figures 6 to 11 below). With a separate code, we calculated the most confused classes for each of the models and we obtained the following top-5 confused classes:

Random Forest: Underground Rap: 480; Dark Trap: 477; Hiphop: 288; RnB: 265; Rap: 192
Decision Tree: Underground Rap: 479; Dark Trap: 475; Hiphop: 286; RnB: 265; Rap: 192
Logistic Regression: Dark Trap: 533; Underground Rap: 382; Hiphop: 305; RnB: 253; hardstyle: 202
Gaussian Naive Bayes: Dark Trap: 753; psytrance: 75; techhouse: 55; Underground Rap: 518; Hiphop: 364
kNN: Underground Rap: 476; Dark Trap: 467; Hiphop: 285; RnB: 281; Rap: 194
QDA: Dark Trap: 640; Underground Rap: 409; Hiphop: 322; RnB: 232; Trap Metal: 170
LDA: Dark Trap: 605; Underground Rap: 432; Hiphop: 274; trance: 271; RnB: 259
Ensemble: Dark Trap: 520; Underground Rap: 378; Hiphop: 289; RnB: 241; Trap Metal: 167

Interestingly, most of the top classes are in the same order for each model but some models have one or two unique classes on top. We expected different classes for Ensemble since we believed the errors would cancel each other out, however, we got the same classes as the most confused.

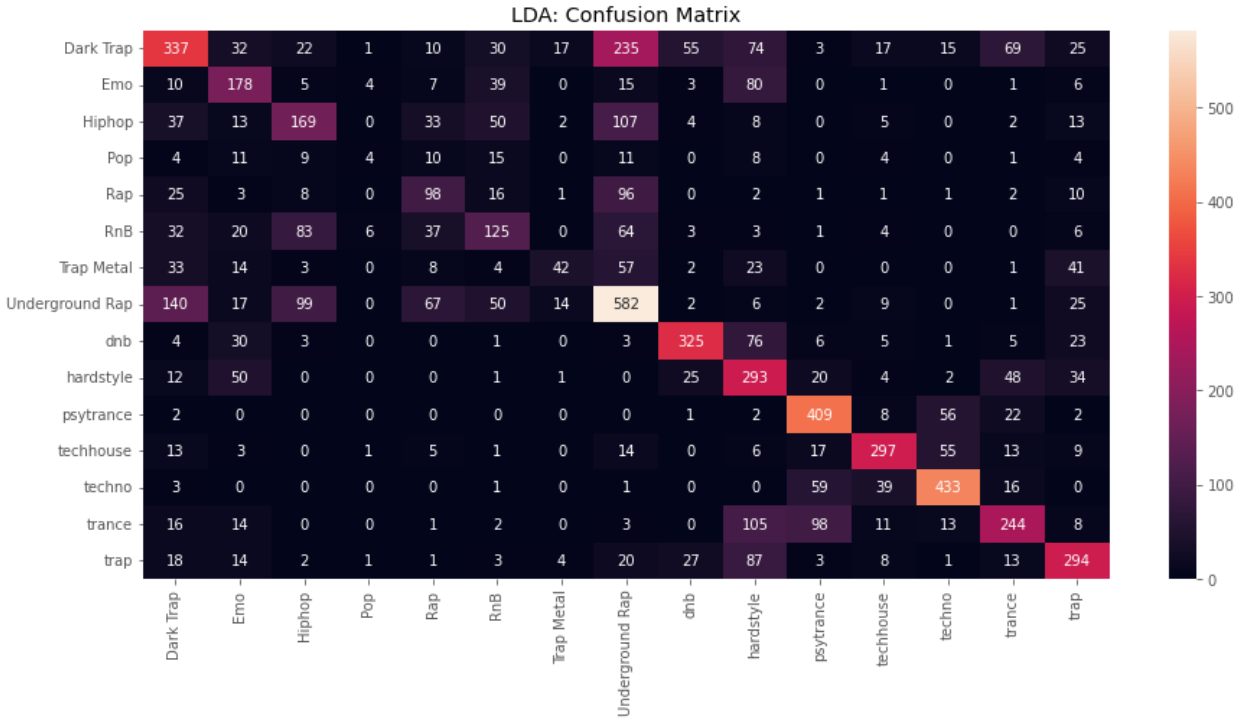

Figure 6. Confusion Matrix for our best performing LDA model.

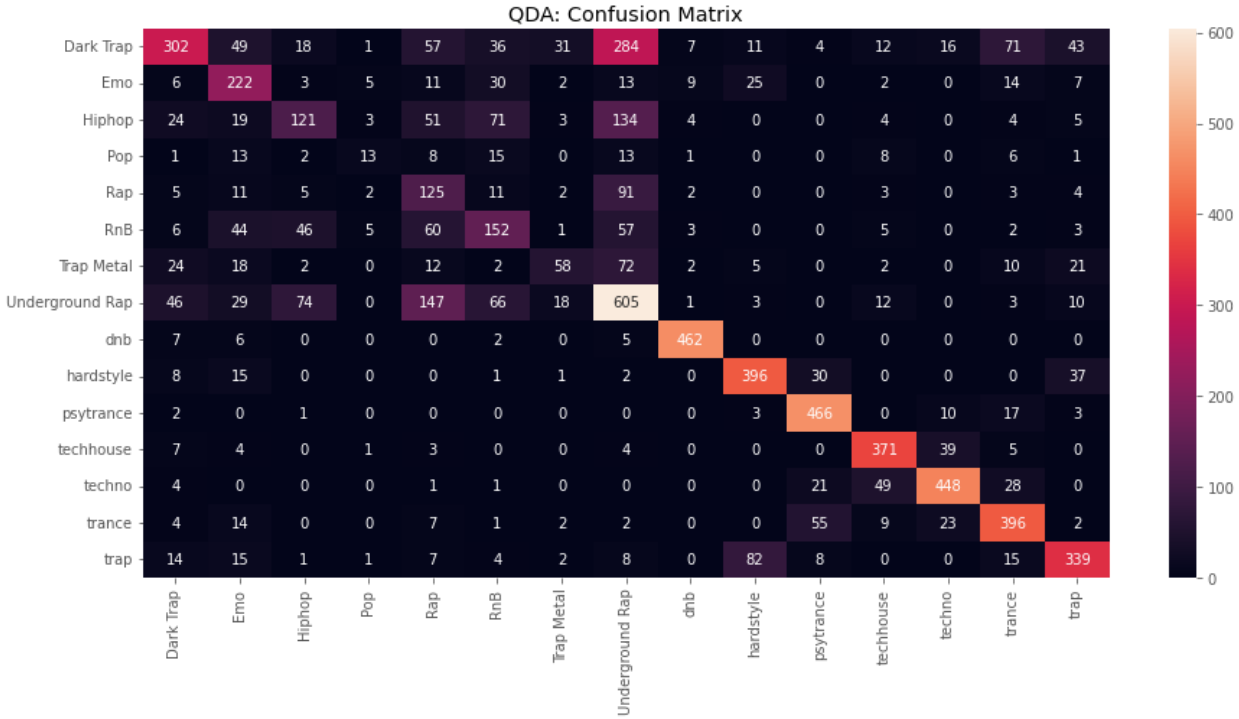

Figure 7. Confusion Matrix for our best performing QDA model.

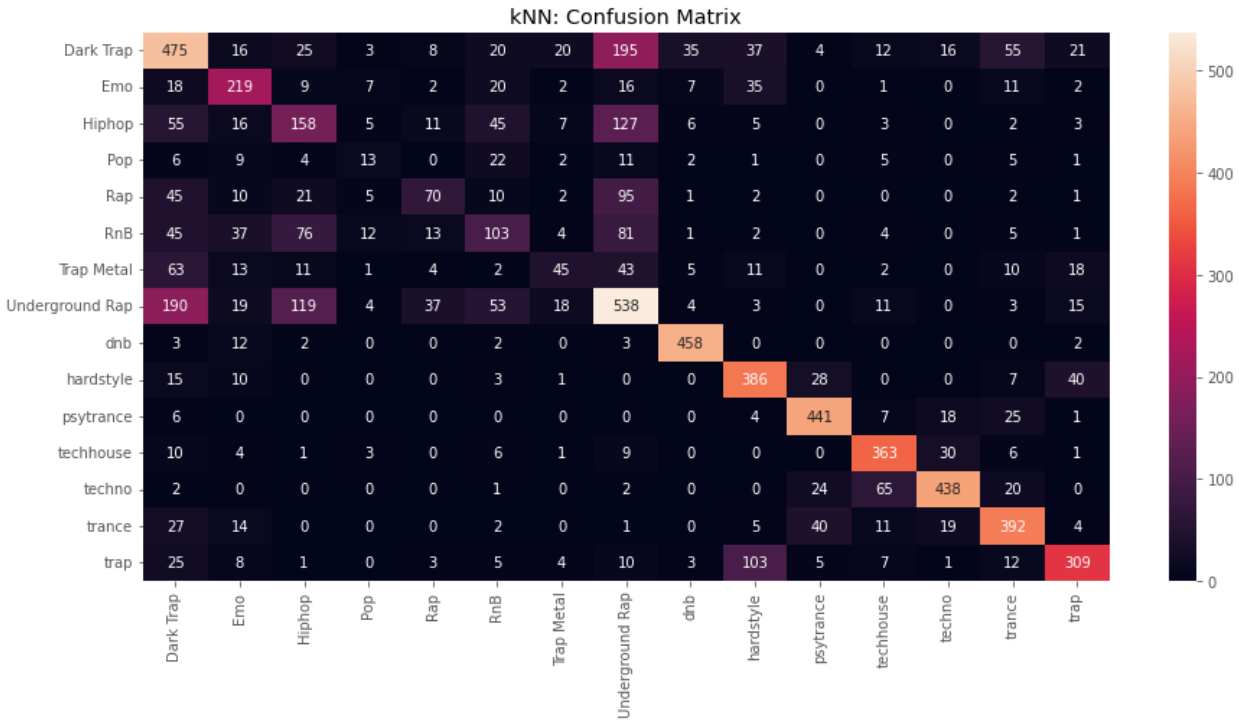

Figure 8. Confusion Matrix for our best performing kNN model.

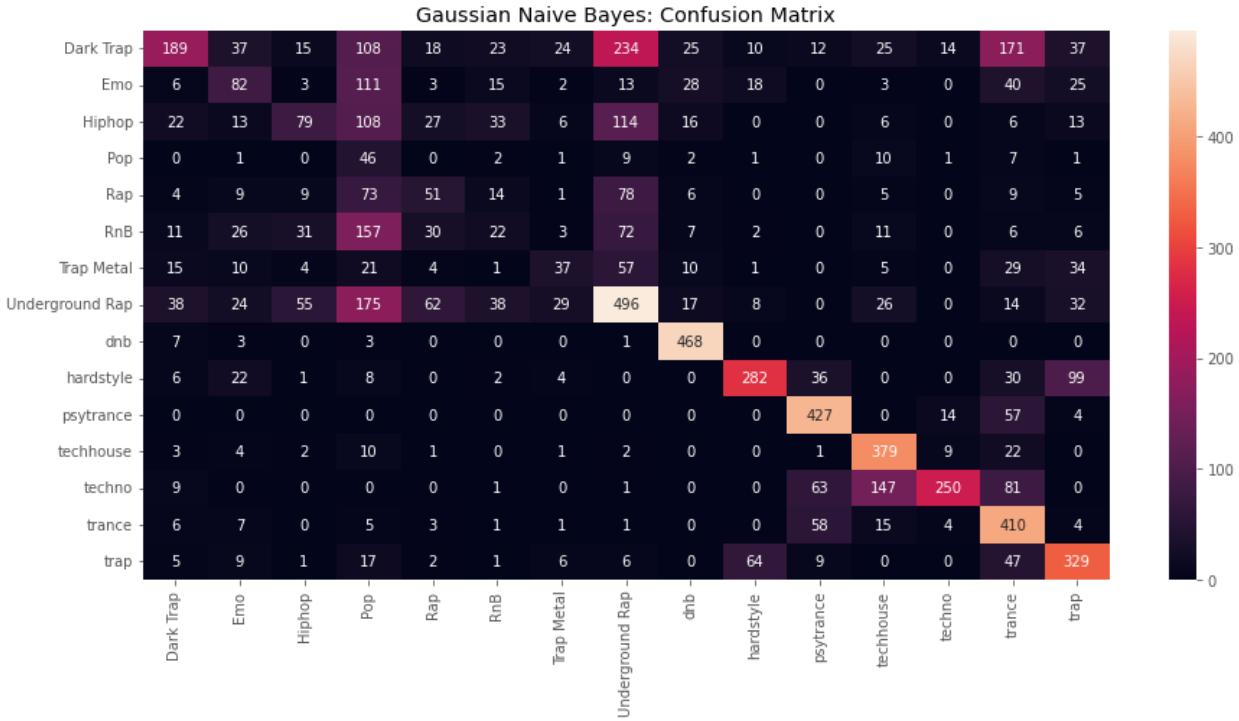

Figure 9. Confusion Matrix for our best performing Gaussian Naive Bayes model.

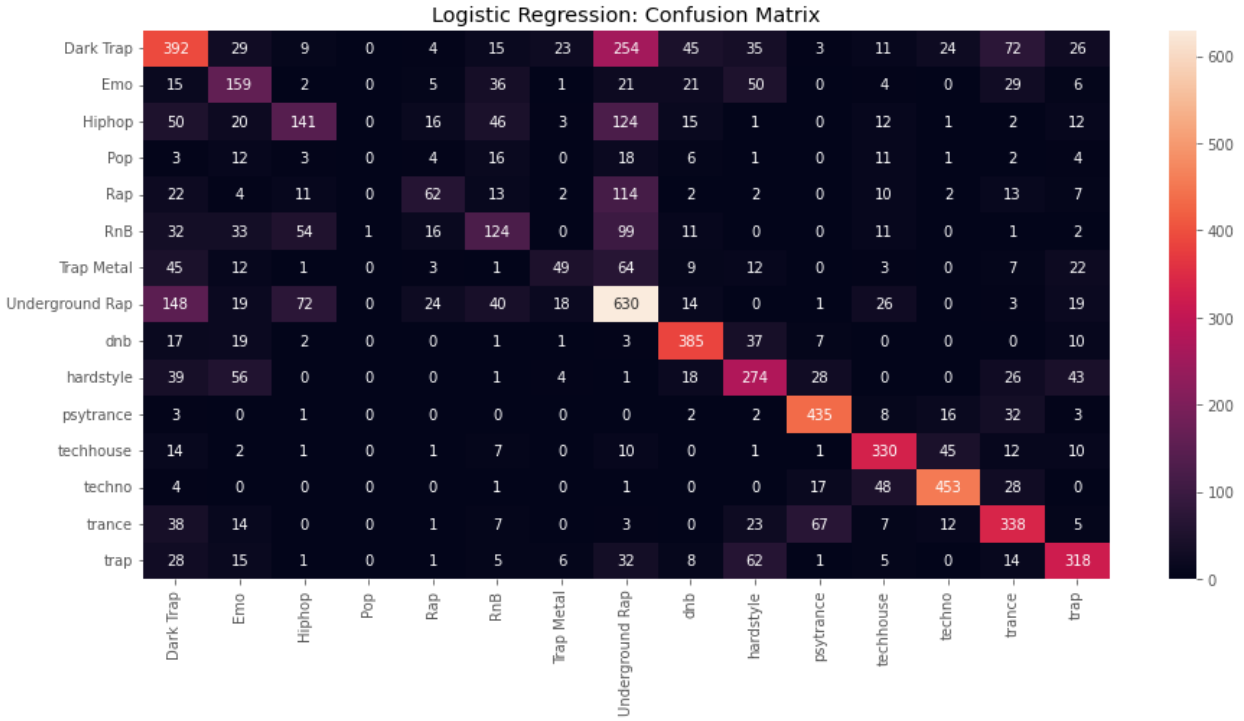

Figure 10. Confusion Matrix for our best performing Logistic Regression model.

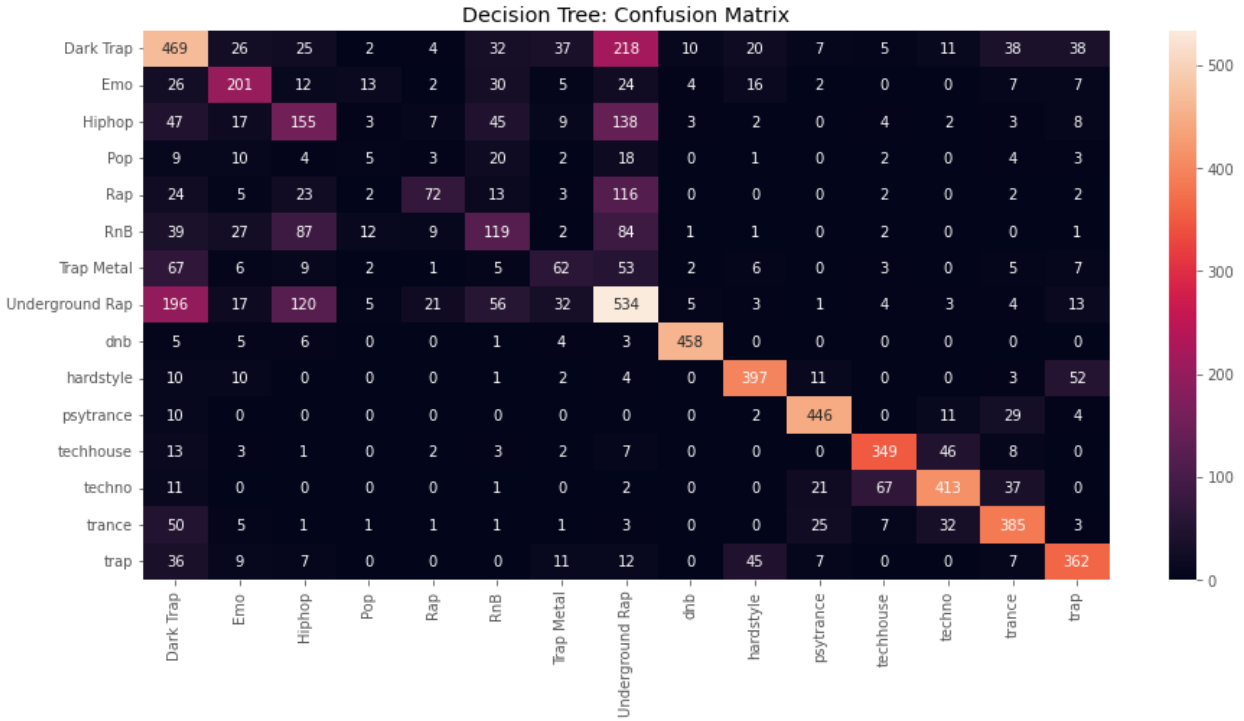

Figure 11. Confusion Matrix for our best performing Decision Tree model.

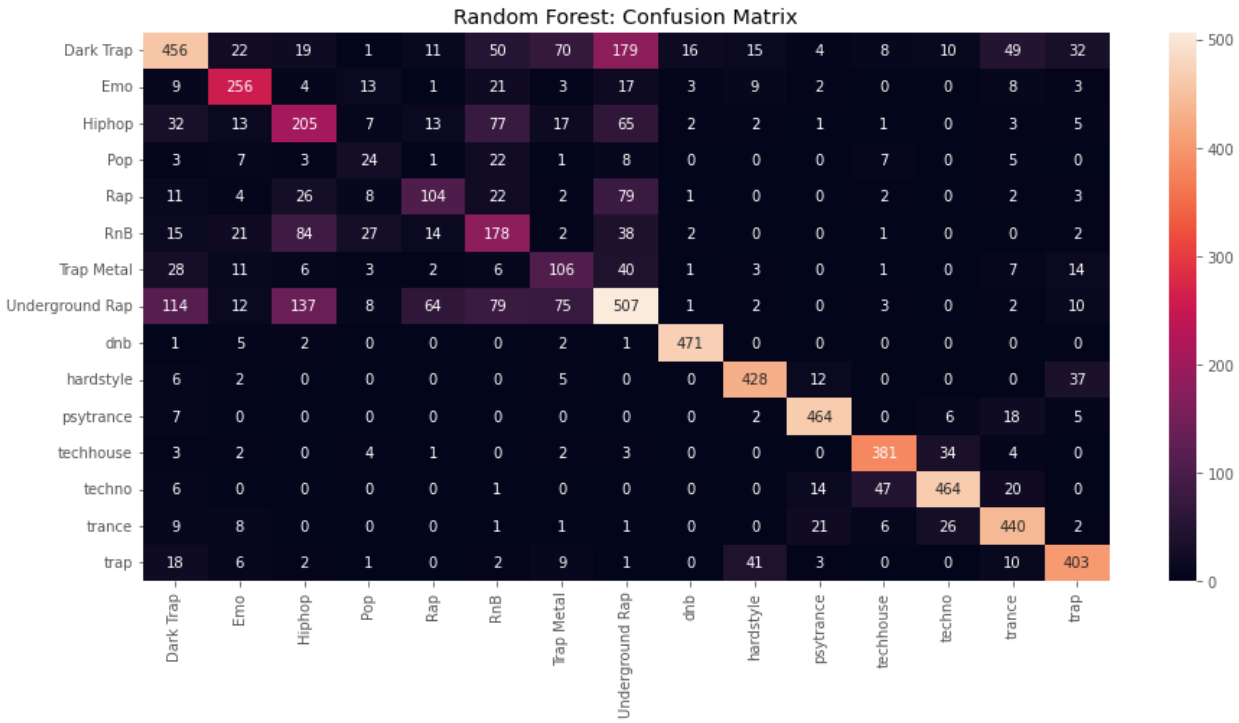

Figure 12. Confusion Matrix for our best performing Random Forest model.

Final Results

Based on the results obtained in the chapter above, the best performing model for this problem and dataset is Random Forest Classifier with 10 most important features. It has an accuracy on test data of 68.1%. After testing again the validation data accuracy, we obtained 68.6%, which is close enough to the test data accuracy. Therefore, we can assume that our model would generalize well on new data. The confusion matrix for the final model can be seen in Figure 12 above.

Conclusions

Personal conclusion

Through this project, we gained a deeper understanding of the complexities involved in classifying music genres. In the first part, data exploration, the key attributes were identified and missing values were handled appropriately. Feature selection and extraction played a significant role in identifying the most important features for classification and we managed to explore the features that make each genre unique.

Scientific conclusion

In summary, Random Forest Classifier was the most effective model for this classification task, with an accuracy of 68.1%. The performance of the models varied, with QDA, Decision Tree, kNN, Logistic Regression, LDA, and Gaussian Naive Bayes also producing acceptable results, but with lower accuracies. The ensemble approach, using a majority voting ensemble, did not yield better results than individual models.

Through the evaluation of confusion matrices, common trends in misclassified genres were observed. This insight highlights the need for further improvement, particularly for the genres that are consistently confused.

Future Work

To further create a more accurate and robust model for the multi-class classification of the Spotify songs, we propose the following future steps:

1. Incorporating a feature importance function specific to each model used. Some features can perform better when used with different models and thus leading to better performance for the models.
2. Leveraging background knowledge to extract relevant features could further improve the classification performance. Individuals with expertise in music could compose features that better capture nuanced patterns of music genres.
3. Creating models for poorly performing classes could further improve the performance. As mentioned above, several classes performed consistently badly. Unique models that are designed to address their unique characteristics could improve the classification accuracy.

Citations

1. Samoshyn, A. (2021). *Dataset of songs in Spotify*, Version 1. Retrieved May 14, 2023 from https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify.
2. Brownlee, J. *A Gentle Introduction to k-fold Cross-Validation*, Retrieved May 20, 2023 from https://machinelearningmastery.com/k-fold-cross-validation/
3. Brownlee, J. (June, 2020). *How to Avoid Data Leakage When Performing Data Preparation*. Retrieved May 20, 2023 from https://machinelearningmastery.com/data-preparation-without-data-leakage/
4. Zebra Keys. *12 Keys of Music*. Retrieved May 20, 2023 from https://www.zebrakeys.com/lessons/beginner/musictheory/?id=12/
5. Signature Sound. *Time Signatures in Popular Music*. Retrieved May 20, 2023 from https://signaturesound.com/timesignatures
6. Scikit-learn. (2023) sklearn.linear_model.LogisticRegression. Retrieved 7 June 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
7. Wang. Y., Cross Validated, *Why do we sample from log space when optimizing learning rate + regularization params?*. Retrieved 8 June 2023, from https://stats.stackexchange.com/q/414227
8. Scikit-learn. (2023) sklearn.naive_bayes.GaussianNB. Retrieved 10 June 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
9. Brownlee, J. (June, 2020). How to Develop Voting Ensembles With Python, Retrieved June 10, 2023 from https://machinelearningmastery.com/voting-ensembles-with-python/