FMA DATATSET, an Exploration
Data Mining 2 2020/21

Marianna Abbattista[1], Fabio Michele Russo[2], and Saverio Telera[3]

Contacts: [1]m.abbattista1@studenti.unipi.it, [2]f.russo11@studenti.unipi.it, [3]s.telera@studenti.unipi.it

June 17, 2021

Contents

1 Data understanding and preprocessing

1.1 Introduction

This is a study on the **Free Music Archive** (FMA), an open and easily accessible dataset suitable for evaluating several **MIR** tasks, a field concerned with browsing, searching, organizing and extracting information from large music collections. For our task we decide to analyze which kind of Album our tracks are, using the dataset `tracks.csv`. So with this task in mind we start by understanding how our big dataset is structured. We have 52 features of which 5 are categorical, 7 are date-time, 2 are float, 15 are integer and 23 are objects; and there's a total of 106574 records.

1.2 Coherence

We found two types of missing values. First, *nan*: this appears in our instances randomly, so we assumed these to be normal missing values due to mistakes or omissions in data entry and we decided to treat them in a following part.
The second missing value we found is *-1*. We did a first coherence pass with integrity checks for the non negativity of variables like *favorites*, *listens* and *comments*, which would not make sense to be negative. We found that there are many instances of these being -1, which we figured to be **missing values** based on the fact that there's a **very high correlation** of -1 appearing in one field and -1 appearing in multiple fields.
This second kind of missing value, *-1*, was replaced with *no info*. We want to treat as data point this kind of absence of values, since we think there's a high chance of these missing values indicating things like those "-1" fields not applying in those specific songs cases (e.g.: a field of artist comments in a song uploaded in a website where it was not possible to add comments to the artist). Also, we chose this approach because this second type of recurring missing value appears in many thousands of our instances.

1.3 Discretization

For some classification methods we will use decision boundaries must be drawn, and where that happens, it makes sense to limit the complexity of those boundaries by discretizing continuous variables, like *listens*. We're not really interested in if a song had 81 or 84 listens; we might be interested instead if listens were, compared to their distribution in this dataset, *very low*, *low*, *medium* or *high*.
Therefore, in an effort of getting the best possible **generalized performance** and minimizing possible sources of overfitting, we chose to discretize several variables. We did two types of discretization:

- discretization of features that we didn't want to keep track of their value but just considering **if there was a value** or **not** (extracting dummy variables);
- discretization of features according their **distribution**.

Using the first method we discretized: *(album, comments)*, *(artist, comments)* and *(track, comments)* in "*no info*" if there was -1 (like previously explained), "*no comments*" if there was 0 and the rest of instances with value "*commented*".
According to their distribution we discretized: *(album, favourites)*, *(artist, favorites)*, *(track, favorites)*, *(album, listens)*, *(track, listens)* in: "*no info*" if there were a -1 value and the rest of instances in "*low*", "*medium*", "*high*", "*higher*" and "*super*".
We also initially discretized *(track, duration)* in "*1min*" if the track is shorter than 60 seconds, "*2min*" if the track is shorter than 120 seconds and "*3min+*" if the song is longer than 120 seconds, but later on we decided to exclude this in the classification task because most classifiers didn't see their performance improved by using it.
For the date variable we decides to only take the year, again, to avoid being overly specific in our data and avoid overfitting.
Of course, this discretization was only applied when useful and it will be specified in each section (e.g.in knn methods, it wasn't used).

1.3.1 Very low coverage attributes

There are a few attributes with very low coverage ($<$10 %) that are very interesting. These are:

- *(artist, active year end)*
- *(artist, wikipedia page)*
- *(track, composer)*
- *(track, date recorder)*
- *(track, information)*
- *(track, lyricist)*
- *(track, publisher)*

Of these, let's take *engineer*. It's not strictly relevant for us the specific name of the sound engineer. We're not going to extract information out of the names, like experience of the professional or their hourly rate. However, the presence of an engineer in the song metadata (which we hope is as complete as song metadata can be) might indicate a recording **made with higher quality** or with a **higher budget** than one without a sound engineer. Therefore we figured that all we cared about, instead of people names, were the **presence or not** of an engineer. Therefore we made dummy variables out of attributes such as this one.
Dummies have value 1 when the attribute was present, 0 when it was missing.

1.4 Other missing values

We made an assumption where the language of title of a certain tracks is the same of the lyrics, so we replaced all missing values of variable *(track, language code)* using a language detector applied on the *(track, title)*, that has full coverage of values instead.

For the rest of variables we decided to simply remove all the rows with missing values since they were relatively very few. Features involved in this process are:

- *(artist, date created)* with 103045 non-null values
- *(track, license)* with 106487 non-null values
- *(album, date created)* with 103045 non-null

We did the same thing for the *(album, type)* variable, even if it has a greater number of missing values (*100066 non-null values*), but we wanted to keep our target variable as accurate as possible.

1.5 Special variables study

1.5.1 Album Type

We decide to set as *target class* **Album Type**, to predict which kind of album type the tracks belong to. *(album, type)* is a categorical feature of 5 values:

- **Album**
- **Live Performance**
- **Radio Program**
- **Single Tracks**
- **Contest**

Looking at the class distribution we saw that *contest* has only 14 records in the entire dataset. So we decided to delete them. Below we can see the distribution of the variable and how we encode that for the classification task.

Album-type value	Number of istance	Encode as
Album	86944	0
Live Performance	5005	1
Radio Program	6524	2
Single Tracks	917	3
Contest	14	Deleted

Figure 1: Album Type Enconding

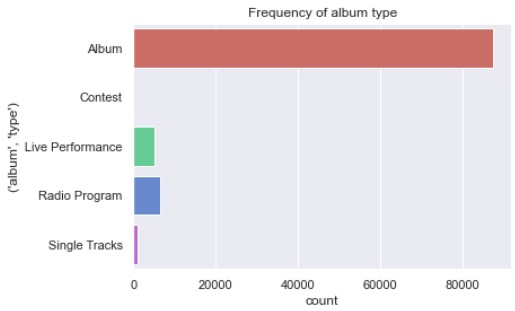


Figure 2: Album Type Distribution

1.5.2 Genres

We noticed that the feature **genre top** had *49598 missing values*, for this reason we decided to fill them with a set of dummy variables. For each genre we added a **dummy** variable that is a binary variable (coded as 1 or 0) to reflect the presence or absence of a particular genre of first level inside the total set of genre per track that we found in the variable *genre all*. Our first level genre variables created are:

- Rock
- Electronic
- Experimental
- Hip-Hop
- Folk
- Instrumental
- Pop
- International
- Classical
- Old-Time / Historic
- Jazz
- Country
- Soul-RnB
- Spoken
- Blues
- Easy Listening

Since some tracks had no first level genre we added a new dummy called **top genre missing** to the set. After the dummy creation we deleted all three features about genres: *genre*, *genre top* and *genre all*.

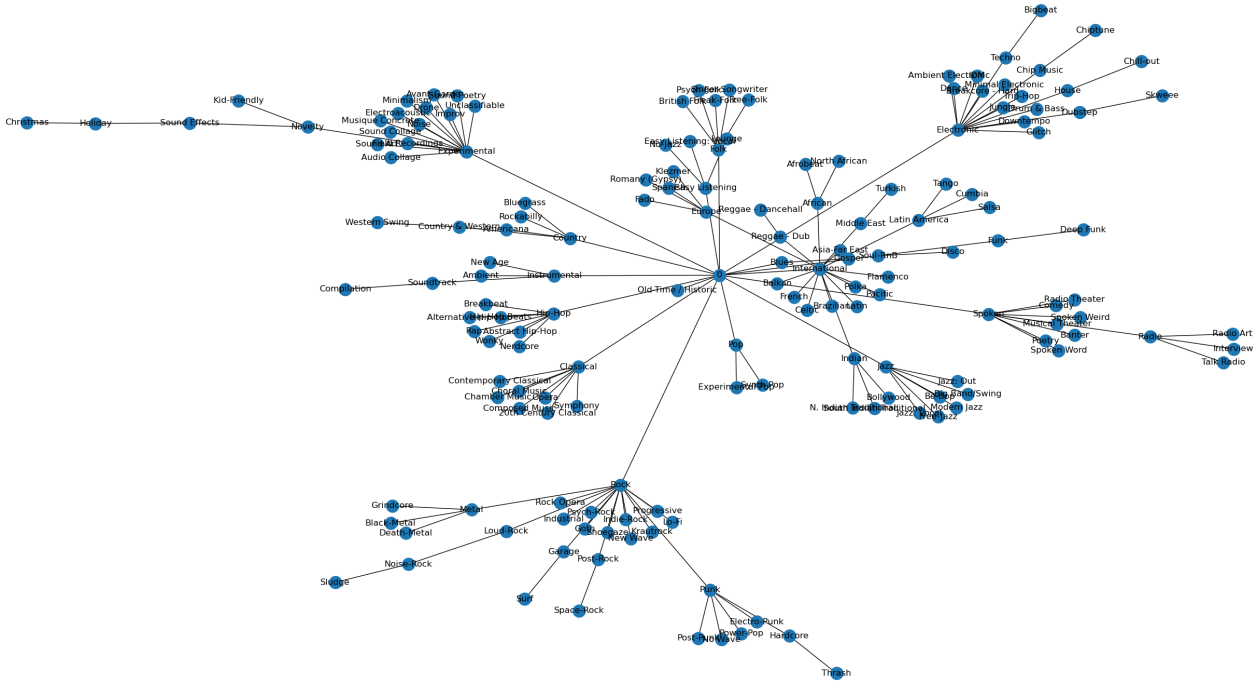


Figure 3: Genres tree

1.5.3 Correlation Matrix

To guide our work, we analyzed and plotted the correlations between attributes with a **Pearson's correlation matrix**. We have as final preprocessed dataset 44 features and 99319 records.

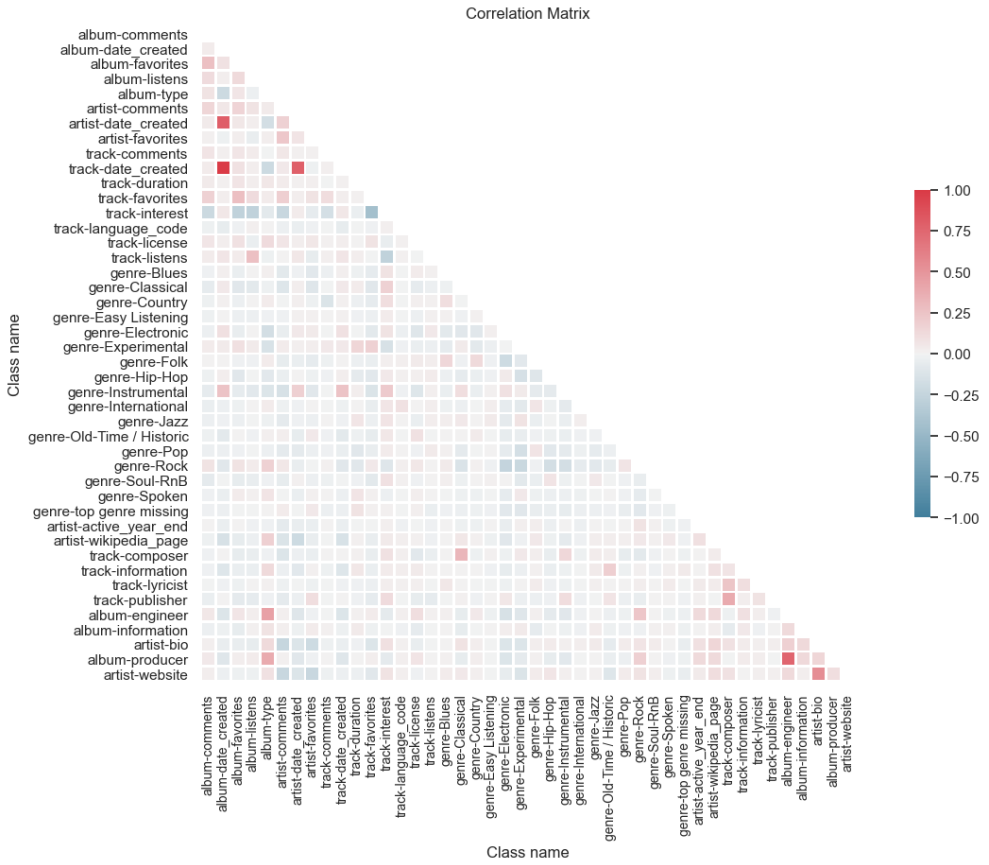


Figure 4: Pearson's Correlation Matrix

2 Starting classification

2.1 Feature selection

We decided to exclude some features in order to avoid any possible undue effects on the classification. For instance, the track number might highly correlate to the track being a single (if ==1) or part of an album (if >1). Moreover, it doesn't make much sense to include the title of the songs or albums in our classification models. We excluded these variables:

- (album, title)
- (album, id)
- (track, title)
- (artist, id)
- (artist, name)
- (track, number).

2.2 Decision Tree Classifier

We started by building a decision tree. We built two different decision trees with same parameters, a first one using discretized features described in paragraph **1.3** and the second one using the continuous features. For the building phase we used 43 features.
After the feature selection our dataset is split using the variable *(set, split)* in our raw data set. A splitting of training and test of 80-20 percent is generated. For the parameters tuning we used a random search with a wide set of different values.

Once found the best set of parameters we run the model. We obtained the following results:

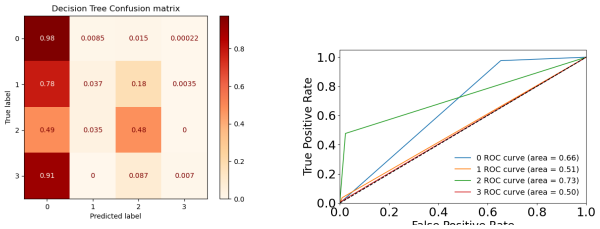


Figure 5: DecisionTree Discrete Features, Confusion Matrix and ROC

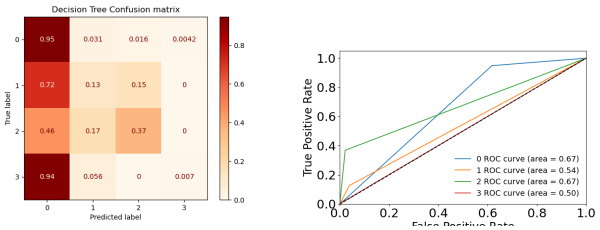


Figure 6: DecisionTree Continuous Features, Confusion Matrix and ROC

As we can see from the results, the decision tree with discretized features is slightly better. As we expected our naive trees tend to classify records as the majority class, artificially increasing the accuracy, which is around **88**%.

Since results are really bad we decided to do a new decision tree using a simple random split but we kept the proportion of the splitting used above with same features discretized since it showed better results. We obtained the following results:

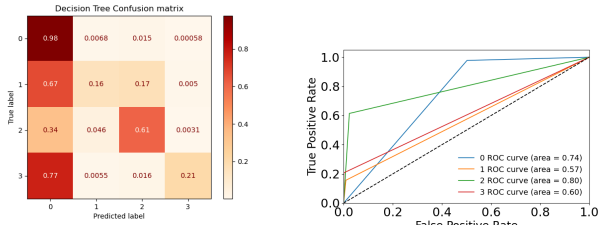


Figure 7: New DecisionTree, Confusion Matrix and ROC

With this new decision tree we obtained better results. We can say that that old division of training and test set penalized the minority class because that splitting was not meant for our target variable, this is overcome using a splitting with stratify option that preserve the proportion.
The feature importance of the top 10 variables used to build the decision trees with discretized features it is been plotted.

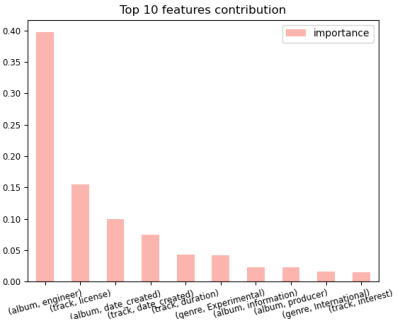


Figure 8: DecisionTree, Feature Importance

As we can see from the plot 8 the most contribution is given by the *(album, engineer)* variables, a Boolean set of values that state if the album has an engineer stated or not.

2.3 KNN Classifier

For this Classifier we decide to use a dataset with **26 features** instead of 44, we do different try to have the best classification set of variable for increasing accuracy and the other metrics.
For any of these variables we treat it because of the fact that not all this variables are integer so we transform all of this into integer value. But not at all using the discretization done during the pre-processing, we decide to leave the continuous variable as it is without discretization, and these are:

- (album, comments)
- (artist, comments)
- (album, favorites)
- (artist, favorites)
- (track, comments)
- (track, favorites)
- (album, listens)
- (track, listens)

Since KNN tunes Euclidean or Manhattan distance through the p parameter, we normalized our data, using **MinMaxScaler**, after preparing the dataset we do the LabelEncoder for encode target labels with value between 0 and n classes-1.
In the **training** phase, for the dataset, we defined stratified splits into 80-20. We performed a 4-folds stratified cross-validation, performing a grid search upon KNN's hyper-parameters.
In the **validation** phase, the best hyper-parameters returned by the grid search were used to make the prediction on the stratified holdouts (validation set).
As for the final **test** phase, scoring the greatest value were used to classify the test set (training, obviously, the KNNClassifier upon the entire training set).
Finally, KNN best models hyper-parameters are the following: **n-neighbors=2, p=1, weights=uniform.**
And the best **accuracy = 0.95%**.
Below are shown the Roc Curve and the Confusion Matrix.

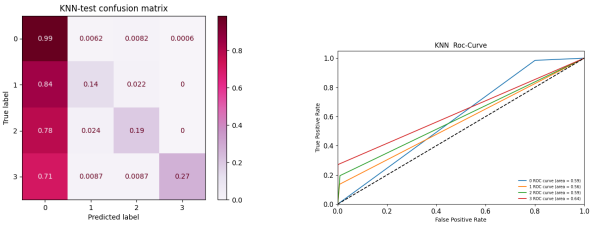


Figure 9: KNN, Confusion Matrix and ROC

2.4 Conclusion

In this chapter we started with a first attempt to classification task of our target variable *(album, type)*.
As we expected both **KNN** and **DecisionTree** have been influenced by the imbalanced proportion of our types of album. Both have high *"Album"* recall, the most frequent values of our variable, since they mostly tend to give in output this label. In fact, all the other classes are less retrieved. This can be seen mostly in the KNN rather than DT, that is negatively effected by the high complexity of dimensionality.
Some feature transformation we did at the beginning, proved to be successful. As we can see from the figure 8, most of the features that gave more contribution to the building model task of our decision tree are *(album, engineer)* and *(track, license)*.

3 Anomaly detection

For this section we decide to adopt different technique from different families. For the Density-based method we adopt **DBSCAN**, for the Angle-based method **ABOD**, for the Model-based approch **Isolation Forest** and for the Distance-based Approch **KNN**. We explore all of them for indetify the top 1% of outliers.

3.1 DBSCAN

For this particular task, our data needs to be normalized, so being remindful that clustering procedure are sensible to noise, we used **StandardScaler** as data normalizer.
After considering different combinations of features, by narrowing or broadening at each trial the considered feature set, we got the best results using the

Euclidean distance as metric and **all the continuous variables**.
For having the top 1% of the outliers we try different combination of minPoints and esp and our best combination is esp = 5000 and minPoints= 24. And at the end we have **1906 anomalies**.

3.2 ABOD

For this method we decide to use the same set of variables used for DBSCAN. For testing our hyperparamenters we test a range of n-neighbor in a range [5,50] and the best results came from n-neighbor= 10, for the contamination we set 0.011 and we have exacltly **1072 anomalies**.

3.3 KNN

Although KNN is a supervised ML algorithm, when it comes to anomaly detection it takes an unsupervised approach, there is no pre-determined labeling of "**outlier**" or "**not-outlier**" in the dataset, instead, it is entirely based upon threshold values. After using **StandardScaler** as data normalizer, we have fitted our NearestNeighbors using the same parameters used for the classification task.
After model building we calculated the distances of k-neighbors and plotted them:

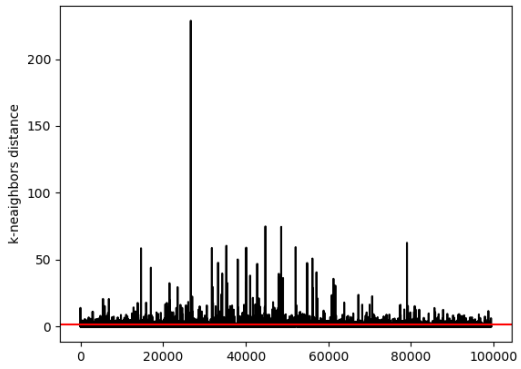


Figure 10: K-NN - Anomaly Detection

After several trials we choose a threshold that catch a number of anomalies closer to our 1 percent of the dataset, as stated by the red line in the plot 10. With a threshold of 3.8 we found **1190 anomalies**.

3.4 Isolation Forest

We trained an isolation forest outlier classification model that is of course an unsupervised model. We can report that we used all continuous attributes in learning the model and could find **7.72%** of the dataset as predicted outliers.

3.5 Choice of method and conclusion

Our general idea starts from the assumption that outliers are points that are at the edges of the dataset's hyperspace; so we decided to pick a statistical measure and evaluate it. We chose the **variance** as this metric.
We figured that for most attributes - like *listens* - some outliers are to be expected when we're dealing with a database of mainly non-commercial music. We decided we mainly wanted to **reduce variance** in this attribute: *(track, interest)*.
Interest is an attribute that has very distinct descriptive statistics across classes. Since this is a mainly non-commercial, free licensed music dataset, anyone can publish music in it, be it professional songwriters or hobbyist musicians. Therefore, we can expect interest to have a very high variance, and it does, particularly in songs in the *Album* category. In fact, standard deviation in the album category is more than x5 than its mean, while in the other minority classes it is only between x1 and x3 of their means. Therefore we calculated the variance of this distribution in the unprocessed dataset and also after eliminating outliers with each method. Of course, we have to balance the reduction in variances we got with the number of instances that we removed, since using each of the three methods led to the removal of a different amount of outliers and any additional data point removed will for sure lead to a decrease (albeit small) in variance. Therefore to account for this we have calculated a delta variance, which we called **var loss per outlier**, which is

$$VarLossPerOutlier_m^a = \frac{var_0^a - var_m^a}{o_m} \quad (1)$$

where var_0^a is the variance of the attribute a before treatment, var_m^a is the variance of attribute a after treatment with method m, and o_m is the number of outliers identified and removed with method m.
Then, we chose the method with the **highest var loss per outlier**, which turned out to be **ABOD**. Seems like that the property of angles to be more stable than distances in high dimensional affected positively the outliers retrival. We removed **1072** outliers that ABOD identified.
Note that this method for choosing an outlier detection algorithm is an approximation itself. The loss in variance as we remove data points is not linear as we have implicitly assumed in our formula, but is in the class of logarithmic functions:

$$VarLossPerOutlier \in \theta(log\ n)$$

because if we imagine removing outliers sequentially from the most outlying to the least outlying data point, the variance decrease will be maximum at the start and minimum in the end. However, ours is an approximation we chose to use to avoid overly complex scaling of variance loss.

Method	Variance	Var Loss	N. Outliers	Var Loss per Outlier
Original	383138,19			
Abod	**16416,86**	**366721,33**	**1072**	**342,09**
Knn	20288,25	362849,94	1190	304,92
Dbscan	10642,36	372485,83	1906	195,43
Isolation Forest	6214,67	377013,52	7631	49,41

Figure 11: Complete results

4 Imbalanced learning techniques

For this task we decide to use the **KNN classifier** because of his results that was a little bit better of the Decision Tree Classifier.We decide to use the KNN without the Top 1% of outliers detected by the **ABOD strategy**. So we use the same columns as before but 98247 record instead of 99319. The metrics of course are the same.So first of all we decide to plot the PCA for the KNN.

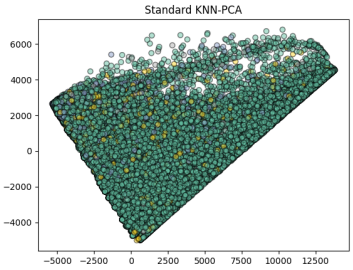


Figure 12: KNN-PCA

So starting talking about imbalance learning we study two principal technique **Undersampling** and **Oversampling** and other way to balance the structure of our target class in the dataset. Reminding that our target class has 4 variables, and is unbalanced, we decide to leave as it is and start playing with the methods. The Target variable from the original data-set shape is divided in: **(0: 64435, 1: 4882, 2: 3743, 3: 678)**

4.1 Undersampling

For the Undersampling technique we propose the **Random undersampling** over the CondensedNearestNeighbour for the best results and we tested the best parameters with a Grid Search and the best are random state=42 and replacement=True. The re sampled variable for the Target class is:**(0: 678, 1: 678, 2: 678, 3: 678)**
After all we plot the new PCA for the dataset as we can see below:

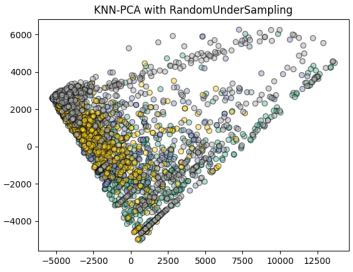


Figure 13: Undersampling PCA

After that we decide to implement this technique into K-NearestNeighbor classifier like we did in the first part ,the Training phase, we defined stratified splits into 80-20 . We performed a 4-folds stratified cross-validation, performing a grid search upon KNN's hyper-parameters , the Validation and the Test phase, all the passage are the same and we have the same results for the hyperparams. Below are showed the Confusion Matrix and the Roc curve.

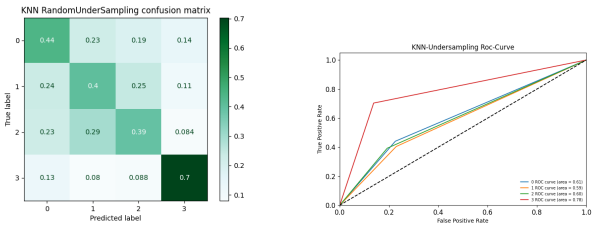


Figure 14: KNN Undersampling, Confusion Matrix and ROC

As you can see this results are vary bad for us classification task, that's enough looking at the accuracy , it decrease a lot,64%!. This signify that for our prediction class the UnderSampling is not the best way to re balance our attributes.

4.2 Oversampling

Let's talk about Oversampling, we perform both **SMOTE** and **Random Oversampling** and we can observe how the PCA changes in this methods. So in this case we tested the best parameters for this two with a Grid Search and the best are random state=42 and replacement=True.
The re sampled variable the Target class became for both: **(0: 64435, 3: 64435, 1: 64435, 2: 64435)**
Here we can observe the two PCA:

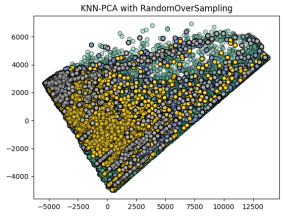


Figure 15: Random OS, PCA

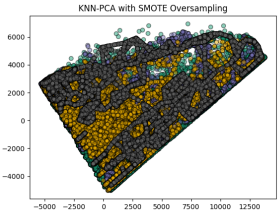


Figure 16: SMOTE OS, PCA

After that ,like in the Undersampling case, we perform the same strategy to apply KNN. This are the final results for both:

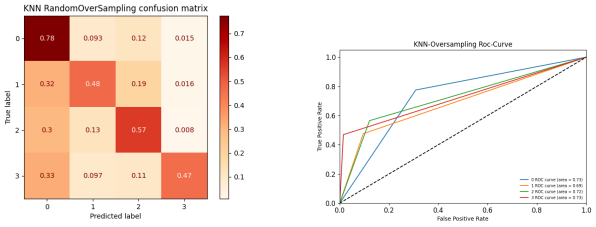


Figure 17: KNN Random OS, Confusion Matrix and ROC

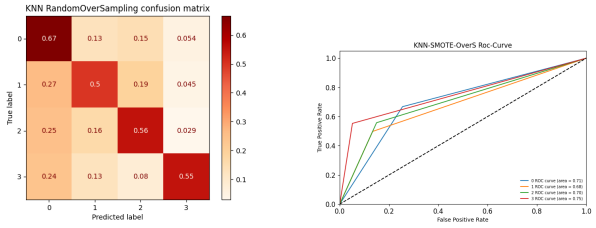


Figure 18: KNN SMOTE OS, Confusion Matrix and ROC

As we can see this two method improves the accuracy, f1 score, ecc... So this means that this method is very nice to adopt for our target class especially SMOTE.

4.3 Weight

For this specific strategy we decide to adopt both Decision Tree an KNN strategy.

4.3.1 KNN

For the KNN we decide to change the weight manually into distance instead of uniform for see if something change and in this case we can see that the performance improve a little bit, but not so much.And we have **overfitting** in the training phase so we can not trust so much in this methods. Here we have the Confusion Matrix and the Roc Curve.

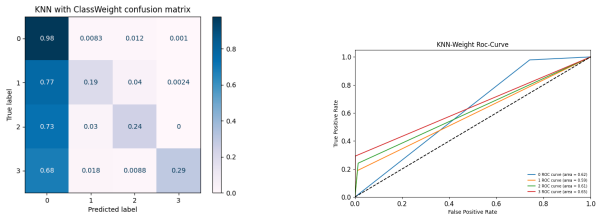


Figure 19: KNN weighted, Confusion Matrix and ROC

4.3.2 Decision Tree

For the Decision Tree we decided to set the weight with the sklearn method **compute class weight**. We executed our model on the training and test set precomputed by the researchers and on a random splitting. Random splitting of training and test showed slightly better results:

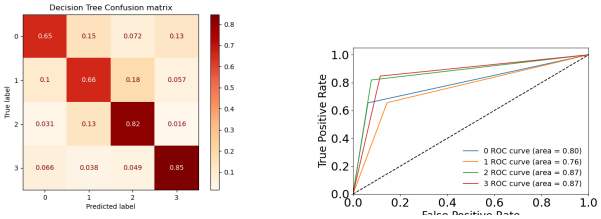


Figure 20: DecisionTree weighted, Confusion Matrix and ROC

As we can see from the results, our classification model increased it's recall, unlike the basic decision tree. This is due to the weight assigned of each class:**0: 0.28, 1: 4.70, 2: 3.60, 3: 31.46** , our classifier put more effort to find the other minority classes but with poor results since our precision is decreased.

4.4 Conclusion

Below we report the table with the total results obtained from this fist part. As you can see for the classification part the KNN and Decision Tree are similar in their general results. The KNN undersampling is the worst of the Imbalace Learnig tecnique and the Random Oversamplig and SMOTE are the best,contrary. Instead for the weighted Decision Tree and KNN changing the weights in distance, we think that KNN is a little bit better but not su much because of the Overfitting of the training.

Model	Parameters	Class	Training - all features				Test - all features			
			Accuracy	F1	Precision	Recall	Accuracy	F1	Precision	Recall
KNN-Base	n_neighbor = 5 , p =1	0	0.905	0.95	0.91	0.99	0.883	0.94	0.9	0.99
		1		0.39	0.79	0.26		0.21	0.49	0.14
		2		0.45	0.83	0.31		0.29	0.6	0.19
		3		0.42	0.89	0.27		0.41	0.83	0.27
DecisionTree-Base (discretized variables) Split Training\Test set precomputed researcher	min_samples_split: 10, min_samples_leaf: 10, max_depth: 9, criterion: gini	0	0.924	0.96	0.94	0.98	0.897	0.95	0.92	0.98
		1		0.45	0.76	0.32		0.05	0.14	0.04
		2		0.69	0.7	0.69		0.5	0.54	0.48
		3		0.47	0.89	0.32		0.01	0.22	0.01
DecisionTree-Base (continuous variables) Split Training\Test set precomputed researcher	min_samples_split: 10, min_samples_leaf: 10, max_depth: 9, criterion: gini	0	0.924	0.96	0.95	0.98	0.87	0.93	0.93	0.95
		1		0.49	0.74	0.36		0.13	0.12	0.13
		2		0.7	0.7	0.7		0.41	0.49	0.37
		3		0.52	0.91	0.36		0.01	0.03	0.01
DecisionTree-Base (discretized variables) Random Split Training\Test set	min_samples_split: 100, min_samples_leaf: 100, max_depth: 8, criterion: entropy	0	0.907	0.95	0.93	0.98	0.905	0.95	0.93	0.98
		1		0.27	0.52	0.18		0.23	0.47	0.16
		2		0.63	0.66	0.61		0.63	0.65	0.61
		3		0.36	0.71	0.24		0.32	0.67	0.21
KNN-RandomUnderS	n_neighbor = 5, p =1, random_state=42,	0	0.44	0.6	0.94	0.44	0.44	0.6	0.93	0.44
		1		0.16	0.1	0.46		0.14	0.09	0.4
		2		0.21	0.14	0.44		0.19	0.13	0.39
		3		0.1	0.05	0.81		0.09	0.05	0.7
KNN-RandomOverS	n_neighbor = 5, p =1, random_state = 42	0	0.86	0.92	1	0.84	0.74	0.85	0.95	0.78
		1		0.65	0.48	1		0.29	0.21	0.48
		2		0.65	0.48	1		0.35	0.25	0.57
		3		0.66	0.49	1		0.31	0.23	0.47
KNN-SMOTEOverS	n_neighbor = 5, p =1	0	0.79	0.86	1	0.76	0.65	0.78	0.95	0.67
		1		0.54	0.37	1		0.25	0.17	0.5
		2		0.6	0.42	1		0.3	0.21	0.56
		3		0.34	0.21	1		0.16	0.09	0.55
KNN-Weighted	n_neighbor = 5, p =1, weights = distance	0	1	1	1	1	0.88	0.94	0.9	0.98
		1		1	1	1		0.27	0.5	0.19
		2		1	1	1		0.34	0.56	0.24
		3		1	1	1		0.42	0.73	0.29
DecisionTree-Weighted	min_samples_split: 10, min_samples_leaf: 10, max_depth: 9, criterion: gini, weight:{0: 0.28, 1: 4.70, 2: 3.60, 3: 31.46}	0	0.65	0.77	0.99	0.64	0.69	0.82	0.96	0.71
		1		0.26	0.16	0.71		0.18	0.12	0.35
		2		0.59	0.49	0.74		0.49	0.39	0.66
		3		0.12	0.07	0.89		0.1	0.05	0.48

Figure 21: Main Results

5 Advanced classification methods

We decided to use a different set of variables (the "complete" dataset) for each classifier, because some require categorical variables and similar concerns. Which ones we used is specified in each method's subsection. Furthermore we decided to run the models with a subset (the "small" dataset) of **specific variables**, in order to evaluate the difference in results given by classification tasks. We chose them, in efforts of improving our classification tasks, using our **feature importance** retrieved from Decision Tree and Random Forest. We tried to include a balanced amount of continue and binary variables. The chosen 10 variables are, for all methods:

- (album, engineer)
- (album, producer)
- (album, information)
- (artist, bio)
- (artist, website)
- (artist,active year end)
- (track, duration)
- (track, interest)
- (track, listens)
- (track, publisher)

We perform classification with these 2 sets of variables, complete and small, and in the final table at the end of the chapter we can observe the performance of both.
For splitting, we did different tries: 60-40, 70-30, 80-20, 90-10 and at the end we decide to use always the same stratified split that is **training=80%** and **testing=20%**. This stratification leads to the best results of our classification methods most of the times and by choosing one type of split only we have results that are comparable with each other.

5.1 Naïve Bayes Classifier

Naïve Bayes (NB) is a probabilistic framework for solving a classification problem. Based on bayes theorem that try to predict an event A, assuming that an event B has occurred. It's assumed that the presence of a particular feature in a class is unrelated to the presence of any other feature.
For the dataset we do different test and at the end we decide to take the same **26 variables** chosen for the KNN and we perform the **LabelEncoder** for encode values between 0 and n classes-1 the classes that needs to be encoded. We observe how the results changes

with different models: Gaussian and Categorical.
In **Gaussian** NB the likelihood of the features is assumed to be gaussian. Is a continuous probability distribution used to describe random variables with real values which tend to concentrate around a single average value.
The results obviously are not very good for our dataset, just the **69%** of accuracy with the complete and the **81%** with the small , of course in Test phase.Seems that the small are better but the other metrics decrease so we can compare and say that this is not a good classification methods for our dataset.
Below are showed the Confusion Matrix and the Roc curve for the complete dataset.

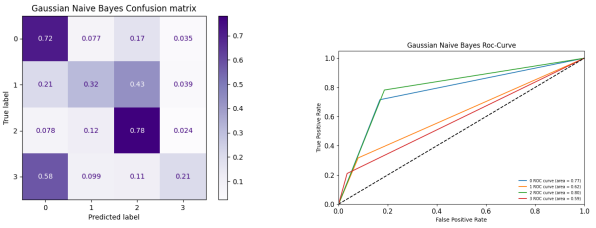


Figure 22: NB Gaussian, Confusion Matrix and ROC
Categorical NB is a classifier for categorical features. It's suitable for classification with discrete features that are categorically distributed. The categories of each feature are drawn from a categorical distribution.
In this case the results increase very much because of the structure of the classifier and we have an accuracy of **91%** in the complete e **85%** in the small.The complete is a very good dataset versus the small that don't classify very well the second and the fourth class.
Below are showed the Confusion Matrix and the Roc curve for the complete dataset.

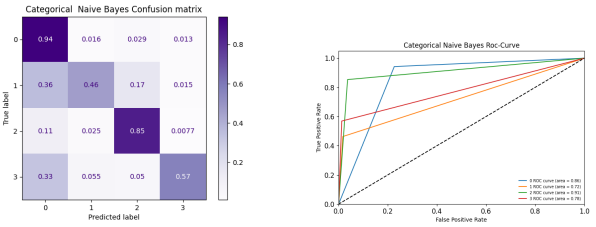


Figure 23: NB Categorical, Confusion Matrix and ROC
Look at the Table 42 for the numerical results.

5.2 Logistic Regression

The logistic regression model itself simply models probability of output in terms of input, by using a cutoff value it is possible to classifying inputs with probability greater than the cutoff decided and below the cutoff as the other; this is a common way to make a binary classifier. For this purpose we decided to **transform** our categorical target variables in a **binary one** with values **Album** and **NotAlbum**.
We decided to keep all our variables continuous so after loading our dataset without outliers and splitting it we fitted the model. To choose our parameters we applied a Grid Search. It gave us as best norm used in the penalization penalty l2, with whom regularization penalizes the Log Likelihood Function with the scaled sum of the squares of the weights; and an inverse of regularization strength equal to 0.1 (smaller values specify stronger regularization).

We built different models and we observed that with normalized features our results were slightly better. Since we do not train our model until the full convergence of the gradient descent, as results, the speed of update of each feature depends on its scale, and when the algorithm stops, some coefficients are closer to the optimum than the others. Feature scaling softens this, because coefficients are now at the same scale and update roughly with the same speed. The implementation of logistic regression uses a penalty on coefficients size L2 norm. In this case, feature scaling matters, because coefficients of features with large variance are small and thus less penalized.

We ordered and plotted the coefficients of the features in a decreasing order in absolute value, to observe in a raw way a sort of feature importance score of our logistic regression model.

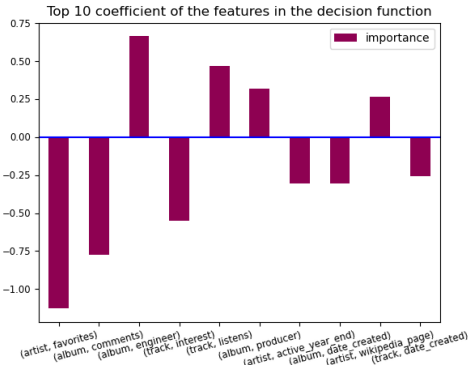


Figure 24: Logistic Regression - Coefficients of features

The positive scores indicate a feature that predicts class 1, whereas the negative scores indicate a feature that predicts class 0. This is an approximation of feature importance. It draws value from the variables being comparable (since we normalized them) but it lacks an important statistical check which would be the significativity test.

For the complete dataset we obtained the following results:

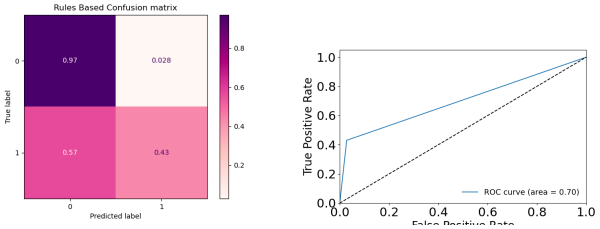


Figure 25: LogisticRegression, Confusion Matrix and ROC

We applied our model also on the our small dataset. As we expected, our results are worst, this is due because some of the variables in the plot 24 that had greater influence on the classification have been removed.

Look at the Table 42 for the numerical results.

5.3 Rule-based Classifiers

A Rule-Based Classifier classifies records by using a collection of "**if...then...**" rules. Since our classification task is multi class we used the following approach:

- We ordered the classes according to increasing class prevalence
- Learn the rule set for smallest class first, treat the rest as negative class
- Repeat with next smallest class as positive class

In our case we followed this order: **Single Tracks**, **Live Performance**, **Radio Program** and **Album** at the end.
We transformed all our categories variables into dummies. A "dummy" variable is a binary variable (coded as 1 or 0) to reflect the presence or absence of a particular categorical code in a given variable. After we our classification task with discretized features.
First of all we applied our RIPPER method for each single class:

- Positive class: Single Tracks precision: **0.80** recall: **0.38** conds: **144**
- Positive class: Live Performance precision: **0.84** recall: **0.27** conds: **709**
- Positive class: Radio Program precision: **0.87** recall: **0.53** conds: **1136**
- Positive class: Album precision: **0.99** recall: **0.62** conds: **701**

Our approach described above is **not mutually exclusive** and **not exhaustive**. In order to manage the fact that a rule might be triggered by more rules we kept all the predictions generated by our 4 set of rules in the previous step and ranked them according to increasing class prevalence. Using this order we assigned to each instance the class label of the highest ranked rule it has triggered. Since a record may not trigger any rules we added as default class the majority class "Album". We calculated our metrics on the entire dataset and obtained the following results:

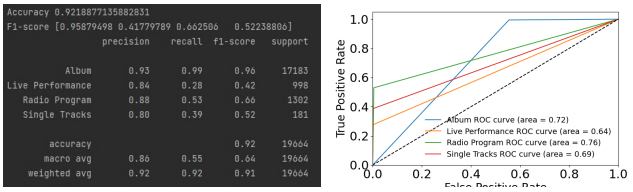


Figure 26: RuleBased, Classification Report and ROC

As we can see our set of rules obtained really high value of precision and low values of recall. Our assumption about the default class increased Album recall from 0.62 to 0.99 with a slightly lower value of precision.
As usual we repeated our classification task on our small dataset.

- Positive class: Single Tracks precision: **0.07** recall: **0.005** conds: **62**
- Positive class: Live Performance precision: **0.43** recall: **0.01** conds: **91**
- Positive class: Radio Program precision: **0.45** recall: **0.06** conds: **881**
- Positive class: Album precision: **0.94** recall: **0.10** conds: **534**

We obtained the following interesting results that we decide to plot only in this case:

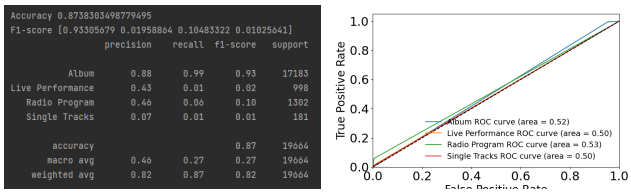


Figure 27: RuleBased small dataset, Classification Report and ROC

As we can see things are definitely worse. The total set of rules went from 2690 to 1568, that as the results showed are not enough to classify our target *(album, type)*.
Look at the Table 42 for the numerical results.

5.4 Support Vector Machines

5.4.1 Linear SVM

For the implementation of Linear SVM we decide to use the dataset with **26 features** so we exclude those variables that semantically dose not provide our scope. So we decide to do a multi class classification with the target class *(album, type)* with 4 values. But with this implementation it was very hard to have good results because of the nature of Linear SVM. So we always split our dataset and do a Grid Search to get the best parameters and these are **C= 0.01**, and this is because of the aggregation of our dataset that has high density, **random state=42**, the better loss function is **squared hinge** and we do max **3000 iteration**. The others remains as they are in default value. Of course we tested different C of our SVM but only with this small value we have an acceptable result, but as you can see in Table 42 it is impossible for us to recognize the last class even though the accuracy is **85%**.
Below are the results of this classification with the confusion matrix and the poor Roc curve.

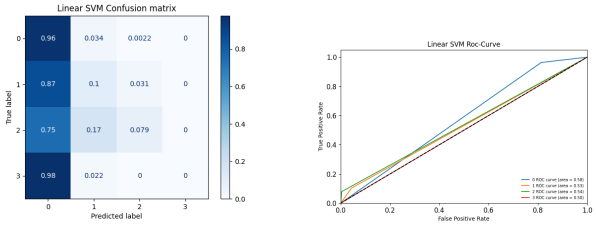


Figure 28: SVM Linear, Confusion Matrix and ROC

So for this problem we decided to **transform** our categorical target variable in a **binary one** with values **Album** and **NotAlbum** to identify only if the track is in an album or not, without more specificity.
For this too we perform a Grid search over the hyperparameter and the results are: **C=100**. This value increases a lot and that is because the target class became binary so the results are more sparse than the other case, **random state=40**, the loss function remains the same, **squared hinge** and this time the best number of iterations to do increases a lot with a max **25000 iterations**. In this case the results improve very well and we have the **89%** of accuracy

and the other metrics says that we have a good chance of finding an album but not a lot to identify very well a non-album (36% of true negative).
Below are the results of Confusion Matrix and you can observe the Roc curve.

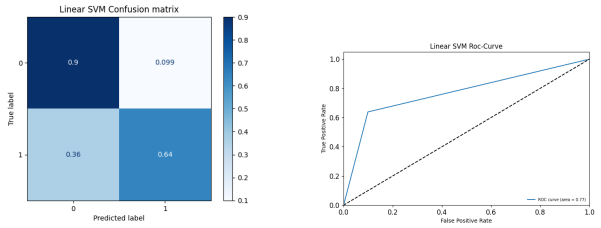


Figure 29: SVM Linear Binary, Confusion Matrix and ROC

5.4.2 Non-Linear SVM

For implementation of Non Linear SVM we used always the same dataset and we tried to use the target variable as it is (all the 4 classes without using the Binary division) hoping that using a non linear kernel function the results improve. And it did, so we set the parameters for the Grid search: **gamma=auto**, **random state= 42**, **probability=true**, **C=0.01**. Then, we decide to loop over different values of kernel function and see how much this decision affects the SVM, so we use both the *rbf* and *poly* kernel functions. Below are the results of these two classifications and, as we can see, the results are quite similar and very good for both. We don't think that one method is really better than the other but as we can see in the Table 42 the **polynomial** kernel function is a little bit better than the **rbf**.

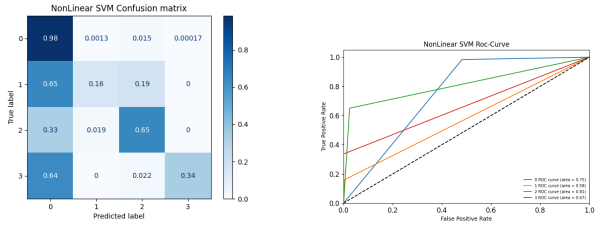


Figure 30: SVM rbf, Confusion Matrix and ROC

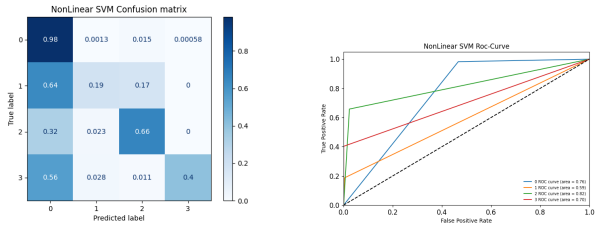


Figure 31: SVM poly, Confusion Matrix and ROC

After that, we decide to prove all of this classification setting with the small dataset chosen by us with the 10 variable using the same Grid search parameters.
For the **Linear SVM** in the case of binary target variable the results improve a little bit mostly the accuracy of the classifier that is 89% instead of 85%, the other metrics change a little but they are quite similar; in the case of multiclass target the results are quite similar and maybe a little bit better because of he recognize the last class but the second one has a very little coverage.
For **Non Linear SVM** instead with the rbf kernel function the result decrease and we can not recognize the last class and the second ones, instead with the polynomial kernel function the results improve vs the rbf but are not better than the poly with all the 26 features.Look at the Table 42 for the numerical results.

5.5 Neural Networks

5.5.1 Single hidden layer

We aimed at building the best single hidden layer neural network first before going on to deeper models.
We started by figuring out which parameters to include in a grid search. As we are not equipped with GPUs, we have to be careful about which parameters to include in our search to balance a good search with our limited computational resources.
We chose these priorities:

- best possible **F1** scores for the minority classes,
- best possible overall **accuracy**.

Therefore, we made the following choices as for parameters to search on and for parameters to leave fixed. Our choices were aimed at spending power on the highest variance hyperparameters possible.

- we tried all four possible *activation functions*: **identity**, **logistic**, **tanh**, **relu**;
- we tried a very rich set of *hidden layer sizes* starting **from 10 up to 500** in variable steps of increasing magnitude (e.g. 10, 20, 30, 40, 50, 65, 80, 100, 120, 150, 180, etc);
- we tried a learning rate initialization value of **0.02**, **0.01**, **0.001**;
- for the *solver* used, we chose the default **adam**, because both in documentation and in our initial neural network explorations it was the one that had the most chances of converging for our large-ish dataset. In fact, none of the other two solvers ever reached convergence in about 90% of our initial explorations. This also gives us the benefit of not having to choose a *learning rate schedule*;
- *max iterations* was left at **200** as it was very rarely used in our actual grid search with **adam**. Most of the times the network reached convergence before *max iterations*.

Overall, we tried **216 single hidden layer neural networks** and we took as best results those within **90.40%** $\pm$ **0.4** accuracy and **88.5 %**$\pm$ **0.5** F1 weighted average, which encompass our top values. Among these, we chose as best the solution with least possible complexity (number of nodes) and which also is within 0.07 of the best weighted F1:
identity, **0.02** learning rate initialization, **40** nodes. **90.46%** accuracy, **89.04%** weighted F1.

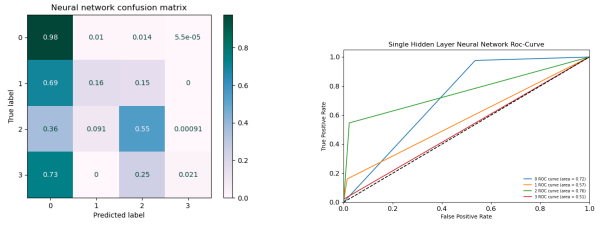


Figure 32: NN, Confusion Matrix and ROC

Another good result we got was the following, which was **very slightly** better in evaluation metrics (it has the top possible F1, which is only 0.07 better than the former choice) but had almost 10 times as many nodes, therefore we would consider the former model better in an effort to reduce complexity and minimize our chances of overfitting.

identity, **0.001** learning rate initialization, **350** nodes. **90.57%** accuracy, **89.11%** weighted F1.

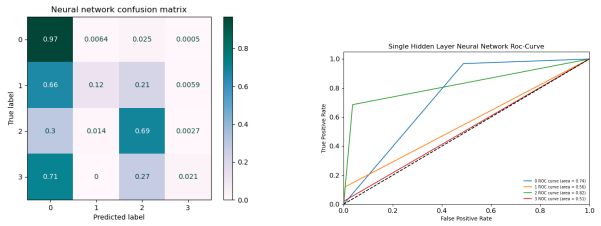


Figure 33: NN, Confusion Matrix and ROC

Having considered these results we started building our deeper neural networks.

5.5.2 Deep Networks

We decided to start from the two best results mentioned above, increase the number of hidden layers and keep the same or reduce the width of each additional hidden layer. We chose this approach once again to better invest our machine power in investigating the results and to try and have our network start from perceiving very general features in the first hidden layer and each subsequent layer start combining the detected features to build more specific ones that can - hopefully - model our class.

We run once again a search, fixing the same hyper parameters as in single hidden layer. We therefore searched on **learning rate initialization** and on **number** and **width of each hidden layer**. Also, on the activation function, but only **identity** and **relu** since they were the only two that appeared in our top single layer results.

We tested **108** deep neural networks. These are the hidden layers' compositions:

(350, 350, 350), (350, 350), (350, 250, 150, 100, 50, 20), (350, 200, 100, 50, 20), (350, 200, 20), (350, 100, 20), (350, 20), (40, 40, 40, 40, 40, 40), (40, 40, 40, 40, 40), (40, 40, 40, 40), (40, 40, 40), (40, 40), (40, 33, 27, 20, 13, 8), (40, 30, 20, 10, 5), (40, 30, 20, 10), (40, 20, 10), (40, 20, 8), (40, 20).

The best two we selected were again the best for overall accuracy and for weighted F1 average. The overall best deep model was with **identity**, **0.001** learning rate initialization and **(40, 40)** nodes in the hidden layers. This scored accuracy=**89.08%** and weighted F1=**86.44%**.

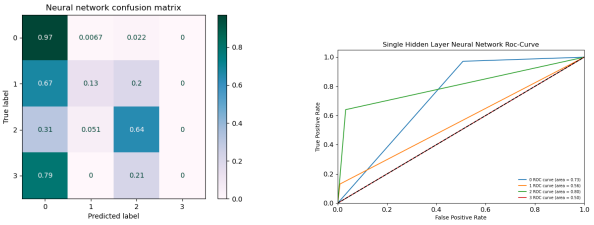


Figure 34: Deep NN, Confusion Matrix and ROC

The second best was one with parameters **identity**, **0.001** and **(40, 20, 8)**. This scored accuracy=**88.97%** and weighted F1=**84.18%**.

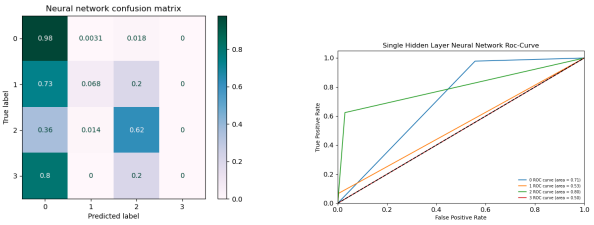


Figure 35: Deep NN, Confusion Matrix and ROC

As we can see both have relatively few nodes. Among the top 10 models only one was with a 350 hidden layer, which we see as an indication that there are not so many complex features that can be extracted from our dataset's attributes. This is also indicated by the fact that adding hidden layers did not improve our metrics. Look at the Table 42 for the numerical results.

5.5.3 More considerations

We found that looking at the F1 the addition of hidden layers **did not help** the classification task in a significant way. Moreover, the class 3 **disappeared** from test classification in the deep models. A poor performance might be expected on it seen that it consists of less than 1% of the dataset. However, looking at the metrics and taking the added complexity into account the best among all the neural networks remains the first one, with **one hidden layer only and 40 nodes** with **identity** and **0.02** init learning rate.

We have to take into account that the excellent accuracy and weighted F1 are mostly due to the overwhelming presence of class 1. In fact, let's look at specific metrics of our best neural network. In tests **class 3** scores a respectable **0.75** precision, but a **0.02** recall. The model is most likely classifying as class 3 only a handful of records, so that the **false positives** will be at a historic low, while the false negatives are completely off the charts and the cause of the abysmal recall. Our learned model did not learn so well.

Why so? We can hypothesize a few scenarios.

1. As already mentioned, there may not be complex features or deep links among our attributes that a neural network could extract;
2. The loss function minimization gives too much weight to the majority class, therefore skewing

the model towards correctly classifying that. We could not find avenues to account for this in the algorithm, while we have tried to counterbalance this in the **imbalanced learning** chapter.

3. The task semantically is a particularly difficult one. Without using any meta attributes (like *number of tracks* in the collection or *ID of the track*, which we scrapped in preprocessing) it is really difficult to understand how and why a song should be a single or one of an album.

5.6 Ensemble Methods

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance.
For the project we decide first to do **Random Forest** alone and after **Bagging** with Decision Tree and Random Forest. At the end we do **Boosting** with AdaBoost apply to Decision Tree and Random Forest.

5.6.1 Random Forest

A Random Forest is a meta estimator that fits a number of decision tree classifiers on various subsamples of the data set and uses averaging to improve the predictive accuracy and control overfitting. We decide to do this classifier with all our dataset first so there are **43 features**. So for this method we do a very big Grid Search for having the best possible results with the minimum number of trees implied. So at fist we analyze the number of trees for building the Random Forest. we set the number of possible trees in a range of **(25-50-75-100-150-200)** and we see that at minimum 100 trees is required for having good results.Furthermore we test the major and most important hyperparamenter of the Random Forest and for us the best combination is **criterion=gini** ,a **max depth of 17**, the **min samples split=3**,the **min samples leaf=3**, **max features="auto"**, **random state=10** and at the end the **class weight="balanced"**. So as you can imagine this involves a lot of time but the results are very good for all the classes. The accuracy is about the **94%** and this represent one of best results that we can observe. Below are showed the confusion matrix and the Roc curve for this methods.

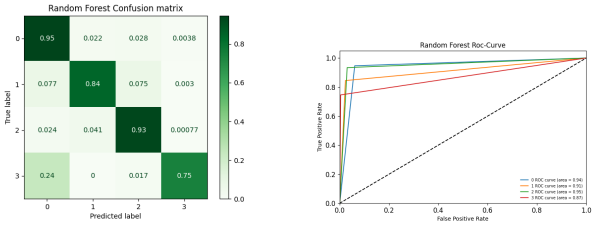


Figure 36: Random Forest, Confusion Matrix and ROC

As we noticed in the previous sections some of the variables we discretized in the data understanding part revealed fundamental for our classification task. We have plotted the Permutation Importance of our Random Forest method and as you can see the top 10-features are the binned variables. Those variables had few coverage and now they occupy the first places of our permutation importance list, this prove us that our first intuition was right and these variables best represent certain conditions that determine an album type.
In particular we can see that if an artist has a website or a bio, an album has a producer, some information or an engineer etc... This kind of information are the most important of our classification task, for example we can see that if a track has a producer generally is always true that this implies a classification in class Album. So contrary if an album has a comments or in base of his date created is not a relevant things to classify him in a certain type of Album and so on looking at the Label 37 below.

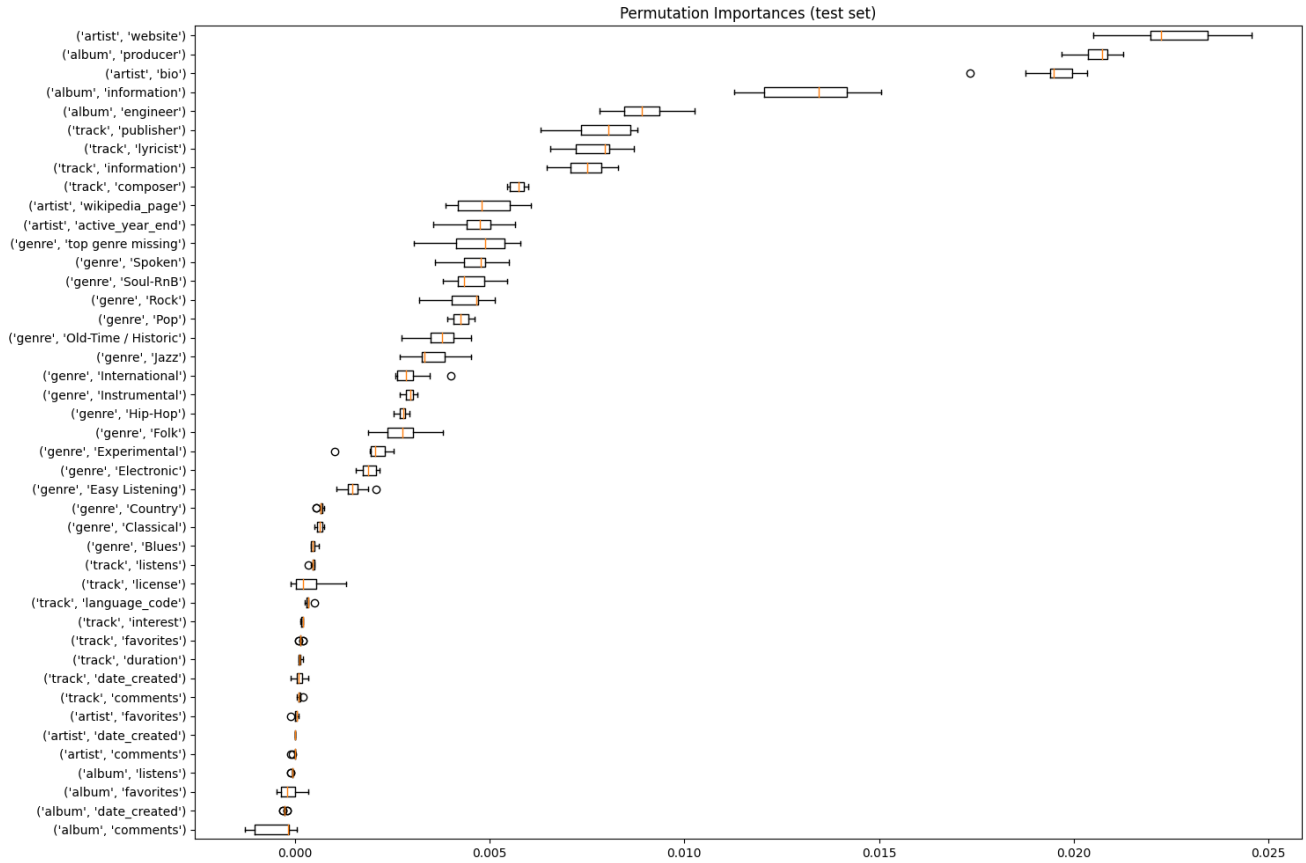


Figure 37: Random Forest Permutation Importance

As you imagine we do the Random Forest with the small dataset basis of this permutation importance and the Decision Tree ones. So for this methods our results using our little set of data are very good so we have the **78%** of accuracy and as you can see from the last table the two class 1 and 3 are poor recognized by the classifier.Look at the Table 42 for the numerical results.

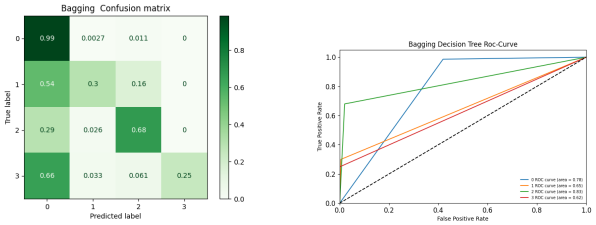


Figure 38: Bagging DT,Confusion Matrix and ROC

For **Random Forest** we do the same configuration of hyperparameters we did before and as you can see the results decrease a little but it's a good classifier still with an accuracy of of **92%**:

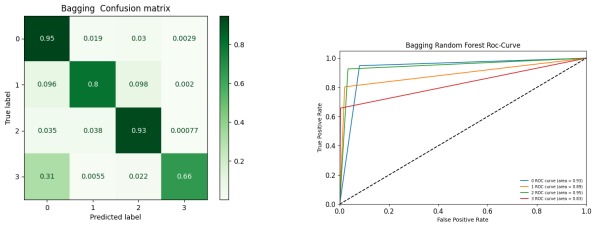


Figure 39: Bagging RF, Confusion Matrix and ROC

We perform the same classifier with the small dataset and the result decrease a little, confirming that this features are the best but we need all the other for having very good results of our classification task.Look at the Table 42 for the numerical results.

5.6.2 Bagging

Bagging, often used what is considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process.

In this project we use two different base estimators to find which works best. We do both with **Decision Tree** and **Random Forest** and we use the same set of complete features with **43 variables**.

For **Decision Tree** we do the same configuration of hyperparameters that we do in the Classification tasks, then **criterion=gini**, a **max depth=9**,the **min samples split=10** and a **min samples leaf=10.**

As you can see the results are worse than the first run. The accuracy instead went up to **92%** even though the 2 classes are decreasing their metrics. Below are showed the results.

5.6.3 Boosting

Boosting, that often considers homogeneous weak learners, learns them sequentially in a very adaptive way (a base model depends on the previous ones) and combines them following a deterministic strategy, but giving different weight to them.
Here we did two experiments where one with **Decision Tree** and **Random forest**. In both cases we observe similar results. The set of hyperparameters is always the same but in training phase of the AdaBoost Random Forest we observe an **overfitting** of the model. So we decide to not take into account this classifier though its results would have been the absolute best.
For both method we can observe that **results are better** rather than our old models. We can observe that precision and recall increased and these two models can now better represent the minority classes due to the best sampling due to boosting.

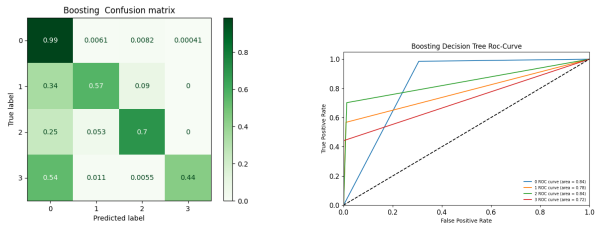


Figure 40: Boosting DT, Confusion Matrix and ROC

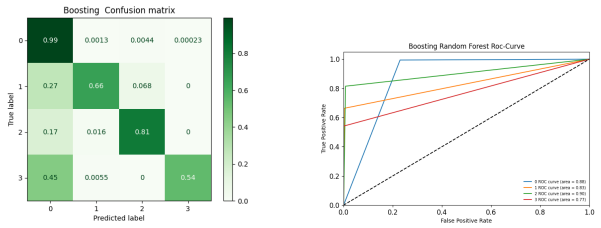


Figure 41: Boosting RF, Confusion Matrix and ROC

We perform the same 2 classifiers with the small set of 10 features and the result decrease a little, confirming that these features are the best but we still need all the other too to get the best results.Look at the Table 42 for the numerical results.

5.7 Conclusion

In conclusion, we have trained "advanced" classifier Gaussian and Categorical Naive Bayes, Logistic Regression, Rule based, linear and nonlinear support vector machine, neural and deep neural networks, ensemble methods (random forest, bagging, boosting).
We have noticed that methods that achieved the higher results are **Random Forest** and the **Rule Based Classifier**.
They both classify instances with same criterion, rules based can be seen as a set of decision or rules, just like a decision tree. We can assume that for our classification aim this is the best way to predict correctly instances of our dataset. These two methods do not consider all variables at the same time to predict our target variable while they are taking decisions, but they follow a certain path (in decision trees or random forest) or certain rules (in rules based) that **might not contain all features**.
So why decision tree weighted did not reach these high results?
We can also notice that when we used bagging and boosting methods on random forest things positively changed. When we act directly on **re-sampling** our instances a classifier have better results.
We introduced this concept in another way in rule based when we defined an order of rules that classify an instance when finding the highest ranked rule it has triggered.
These things were not included in our decision tree weighted because it just increased their recall without any benefit for our task.
Following this reasoning, methods that for definition put at the center of analysis the distribution of classes, such as a probabilistic framework like Naive-Based classifier, did not obtained same results, why?
When we applied it on the entire set of variables results were lower than them obtained on our 10 features selected. These pruning step increased it because as we said at the beginning just a few set is really required for our task, also because correlated attributes can degrade the performance of our Naive Bayes Classifiers.
And what about methods that for definition put at the center of analysis similarity between instances such as Support Vector Machine? With these methods we have different results like we see in previous part. The Linear SVM of course are very bad because they are based on distance between a line and only with a Binary Class they work well. Instead for the Non Linear SVM the situation changes, we can select the Kernel function and using that for the multi-class the results increase very well but not so much to get as good as the results of our best models.
So why on the smaller set of 10 features the random forest or rules based performances did not increase?
It's likely because during feature selection for the 10 we ended up reducing the likelihood of finding the best combination of variables to classify the album types.

Model	Parameters	Class	Training - all features				Test - all features				Training - 10 features				Test - 10 features			
			Accuracy	F1	Precision	Recall	Accuracy	F1	Precision	Recall	Accuracy	F1	Precision	Recall	Accuracy	F1	Precision	Recall
Naïve-Bayes (Gaussian)		0	0.7	0.82	0.97	0.72	0.69	0.82	0.97	0.72	0.81	0.9	0.94	0.86	0.81	0.9	0.94	0.86
		1		0.21	0.16	0.29		0.23	0.18	0.32		0.14	0.15	0.13		0.14	0.15	0.13
		2		0.36	0.23	0.78		0.35	0.23	0.78		0.49	0.37	0.74		0.5	0.37	0.75
		3		0.08	0.05	0.19		0.09	0.05	0.21		0.06	0.04	0.11		0.04	0.03	0.08
Naïve-Bayes (Categorical)		0	0.93	0.97	0.97	0.96	0.91	0.95	0.97	0.94	0.88	0.94	0.93	0.95	0.85	0.93	0.92	0.94
		1		0.71	0.75	0.66		0.52	0.59	0.46		0.22	0.66	0.13		0.04	0.11	0.03
		2		0.78	0.69	0.89		0.72	0.62	0.85		0.52	0.44	0.63		0.43	0.37	0.52
		3		0.55	0.47	0.66		0.39	0.3	0.57		0.17	0.69	0.1		0.02	0.11	0.01
Logistic Regression [album, non-album]	C=0.1, penalty='l2'	0	0.904	0.95	0.92	0.97	0.903	0.95	0.92	0.95	0.887	0.94	0.92	0.96	0.886	0.94	0.92	0.95
		1		0.53	0.7	0.43		0.53	0.69	0.53		0.48	0.57	0.41		0.48	0.57	0.41
SVM Linear Binary [album, non-album]	C=100, random_state=42, max_iter=25000, loss=squared_hinge	0	0.87	0.92	0.95	0.9	0.87	0.92	0.95	0.9	0.89	0.94	0.93	0.94	0.89	0.94	0.93	0.94
		1		0.55	0.49	0.64		0.55	0.48	0.64		0.53	0.57	0.5		0.53	0.55	0.5
SVM Linear	C= 0.01,random_state=42, max_iter = 3000,loss="squared_hinge"	0	0.85	0.93	0.89	0.97	0.85	0.93	0.89	0.96	0.84	0.92	0.88	0.96	0.84	0.91	0.88	0.95
		1		0.12	0.13	0.11		0.11	0.11	0.1		0.01	0.25	0		0	0.12	0
		2		0.13	0.58	0.08		0.14	0.6	0.08		0.14	0.49	0.08		0.15	0.51	0.09
		3		0	0	0		0	0	0		0.02	0.01	0.04		0.02	0.01	0.05
SVM NonLinear	kernel = rbf, gamma= auto	0	0.92	0.96	0.94	0.99	0.91	0.96	0.93	0.98	0.88	0.94	0.91	0.97	0.88	0.94	0.91	0.97
		1		0.28	0.82	0.17		0.26	0.77	0.16		0.04	0.61	0.02		0.04	0.59	0.02
		2		0.67	0.67	0.67		0.65	0.65	0.65		0.48	0.48	0.48		0.48	0.47	0.48
		3		0.58	0.97	0.42		0.5	0.95	0.34		0	0	0		0	0	0
SVM NonLinear	kernel = polynomial, gamma= auto	0	0.92	0.96	0.94	0.98	0.92	0.96	0.94	0.98	0.88	0.94	0.91	0.97	0.88	0.94	0.91	0.97
		1		0.32	0.83	0.2		0.3	0.77	0.19		0.02	0.61	0.01		0.02	0.77	0.01
		2		0.67	0.67	0.67		0.66	0.66	0.66		0.49	0.47	0.52		0.49	0.47	0.52
		3		0.67	0.97	0.51		0.55	0.88	0.4		0.02	1	0.01		0.01	1	0.01
Rule-Based Classifier	k=1, prune_size=0.33	0	0.922	0.95	0.92	1	0.921	0.93	0.93	0.99	0.874	0.93	0.88	1	0.873	0.93	0.88	0.99
		1		0.44	0.94	0.29		0.41	0.84	0.28		0.01	0.62	0.01		0.01	0.43	0.01
		2		0.64	0.91	0.5		0.66	0.88	0.53		0.04	0.5	0.02		0.104	0.46	0.06
		3		0.57	0.94	0.41		0.52	0.8	0.39		0.01	0.29	0.01		0.01	0.07	0.01
EC Random Forest	n_estimators=100, criterion="gini", max_depth=17, min_samples_split=3, min_samples_leaf=3, max_features=" auto", random_state=10, class_weight=" balanced"	0	0.95	0.97	1	0.95	0.94	**0.97**	0.99	0.95	0.82	0.9	0.98	0.84	0.78	0.88	0.95	0.82
		1		0.81	0.72	0.92		**0.74**	0.66	0.84		0.35	0.27	0.51		0.17	0.13	0.24
		2		0.83	0.72	0.97		**0.79**	0.69	0.93		0.59	0.45	0.86		0.52	0.39	0.76
		3		0.83	0.72	0.99		**0.7**	0.66	0.75		0.41	0.27	0.92		0.16	0.11	0.33
EC-Bagging (Decision Tree)	criterion="gini", max_depth=9, min_samples_split=10, min_samples_leaf=10	0	0.93	0.96	0.94	0.99	0.92	0.96	0.94	0.99	0.89	0.94	0.91	0.98	0.88	0.94	0.91	0.97
		1		0.42	0.8	0.28		0.43	0.78	0.3		0.07	0.74	0.04		0.06	0.59	0.03
		2		0.71	0.74	0.69		0.69	0.71	0.68		0.48	0.51	0.45		0.45	0.47	0.43
		3		0.44	1	0.28		0.4	1	0.25		0.26	0.89	0.15		0.21	0.88	0.12
EC-Boosting (Decision Tree)	criterion="gini", max_depth=9, min_samples_split=10, min_samples_leaf=10	0	0.98	0.99	0.99	0.99	0.94	0.97	0.96	0.99	0.89	0.95	0.95	0.95	0.83	0.92	0.91	0.93
		1		0.88	0.91	0.85		0.65	0.76	0.57		0.45	0.43	0.47		0.12	0.13	0.12
		2		0.9	0.91	0.89		0.75	0.8	0.7		0.55	0.56	0.54		0.34	0.37	0.32
		3		0.98	0.98	0.97		0.6	0.92	0.44		0.75	0.78	0.72		0.15	0.28	0.1
EC-Bagging (Random Forest)	n_estimators=100, criterion="gini", max_depth=17, min_samples_split=3, min_samples_leaf=3, max_features=" auto", random_state=10, class_weight=" balanced"	0	0.95	0.97	0.99	0.95	0.94	**0.97**	0.99	0.95	0.84	0.92	0.96	0.87	0.82	0.91	0.95	0.87
		1		0.79	0.72	0.87		**0.74**	0.68	0.8		0.34	0.3	0.39		0.19	0.17	0.21
		2		0.8	0.69	0.95		**0.77**	0.66	0.93		0.57	0.44	0.83		0.52	0.4	0.76
		3		0.83	0.76	0.91		**0.67**	0.69	0.66		0.43	0.34	0.58		0.22	0.18	0.28
EC-Boosting (Random Forest)	n_estimators=100, criterion="gini", max_depth=17, min_samples_split=3, min_samples_leaf=3, max_features=" auto", random_state=10, class_weight=" balanced"	0	0.99	1	0.99	1	0.96	0.98	0.97	0.99	0.97	0.99	0.99	0.99	0.86	0.93	0.91	0.95
		1		0.96	0.99	0.94		0.78	0.94	0.66		0.9	0.91	0.89		0.14	0.2	0.11
		2		0.95	0.96	0.94		0.85	0.88	0.81		0.86	0.83	0.89		0.4	0.42	0.38
		3		1	1	1		0.69	0.96	0.54		0.99	0.99	1		0.23	0.54	0.14
Single hidden layer Neural Network	activation: identity learning_rate_inits: 0.02 hidden_layer_size: 40	0	0.89	0.95	0.91	0.99	0.91	0.96	0.94	0.98	0.87	0.94	0.92	0.96	0.88	0.94	0.92	0.96
		1		0.17	0.34	0.11		0.21	0.33	0.16		0.02	0.3	0.01		0.01	0.27	0
		2		0.51	0.67	0.41		0.56	0.58	0.55		0.48	0.42	0.54		0.46	0.4	0.55
		3		0	0	0		0.04	0.75	0.02		0	0	0		0	0	0
Single hidden layer Neural Network	activation: identity learning_rate_inits: 0.001 hidden_layer_size: 350	0	0.89	0.95	0.92	0.97	0.9	0.95	0.94	0.97	0.87	0.93	0.88	1	0.89	0.94	0.9	1
		1		0.13	0.38	0.08		0.19	0.44	0.12		0	0	0		0	0	0
		2		0.57	0.57	0.57		0.59	0.52	0.69		0.18	0.54	0.11		0.17	0.45	0.1
		3		0.01	0.03	0.01		0.04	0.26	0.02		0	0	0		0	0	0
Deep Neural Network	activation: identity learning_rate_inits: 0.001 hidden_layer_sizes: 40, 40	0	0.89	0.95	0.92	0.98	0.91	0.96	0.94	0.97	0.87	0.94	0.9	0.97	0.89	0.94	0.92	0.97
		1		0.15	0.35	0.1		0.2	0.38	0.13		0.01	0.28	0.01		0	1	0
		2		0.55	0.6	0.5		0.58	0.53	0.64		0.44	0.46	0.42		0.43	0.43	0.43
		3		0	0	0		0	0	0		0	0	0		0	0	0
Deep Neural Network	activation: identity learning_rate_inits: 0.001 hidden_layer_sizes: 40, 20, 8	0	0.89	0.95	0.92	0.98	0.91	0.96	0.93	0.98	0.87	0.93	0.87	1	0.89	0.94	0.89	1
		1		0.11	0.41	0.07		0.12	0.45	0.07		0	0	0		0	0	0
		2		0.56	0.61	0.51		0.59	0.55	0.62		0.09	0.47	0.05		0.06	0.34	0.03
		3		0	0	0		0	0	0		0	0	0		0	0	0

Figure 42: Main Results

6 Linear regression

6.1 Univariate problem

We decided to try and predict how many favorites a track will have based on how many listens it has. Therefore our regression problem is:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where x is *listens*, y is *favorites*, the betas are **coefficients** and ϵ is the **residual**. We found excellent evaluation metrics considering that there's only one regressor, i.e. a very high **53%** R^2. (other metrics at the end)

However, we have to also recognize that this problem, as likely many other regression problems we could have chosen on this dataset, suffers from a **reverse causality** issue. Not only does x influence y, but as the number of favorites a track has does influence the **recommendation** algorithm and how often that track is suggested to users who haven't listened to it yet,

favorites also influence listens, that is y influences x. Therefore we cannot be sure that our regressor **correctly** captures the causal effect of x on y and the best we can do is underline the high covariance that these two variables capture of each other and avoid drawing other, stronger conclusions.

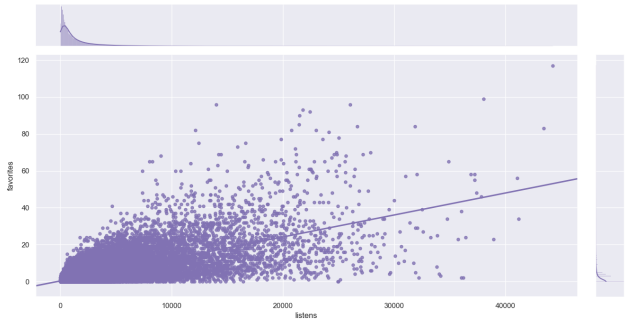


Figure 43: Linear univariate regression with scatterplot of datapoints and distribution curves

The graph shows a particularly **imbalanced distribution** of both of these variables, with one large high peak in the low values and relatively very few points in the top 2 quartiles. **OLS** regression is centered around the assumption of a **normal** distribution, which in this case, it's far from the truth. Therefore we should consider this as one of the problems warning us of the poor **predictive** capability of this regression function, notwithstanding the apparently good metrics.

6.2 Multivariate problem

The multivariate problem is the same effort of predicting *(track, favorites)* but using as regressors all continuous and binary variables instead of just one. This method and its results lead to some interesting considerations.

We got a **63.8 %** R^2 with a **7.600** MSE and **1.570** MAE.

First of all, this result is generally better in its assumed predictive capability than the univariate one, seeing as the metrics are better. However, we also have to consider two issues:

1. The addition of **24 regressors** only increased the R^2 by **20%** of the previous one. This also suggests an issue, just as much as it did that in the univariate problem only one variable could account for a 53% R^2.
2. The reverse causality issue in the univariate is getting bigger here in the multivariate. Anything that is done during music production could not be affected by how many favorites the track has after "commercial" release. However, we also have our three *comments* features that might be influenced by *favorites* in the same way that *listens* is: the recommendation algorithm suggests a song more as it has more favorites. However, this issue should be smaller in entity than that related to *listens*, as we believe the listens to be the main driver of the recommendations. Also, the reverse causality problem could be diluted as there are many features. Therefore, our **63.8%** R^2 could be closer to a genuine, issue-free R^2 than that of the univariate.

In conclusion, we would not use regression to predict any of these variables. There are too many unsolved doubts about distribution, reverse causality, possible omitted features and how the recommendation algorithm works. **OLS regression** requires strong assumptions in these fields and we're not confident we can make them with no negative repercussions. But, if forced to solve this problem with regression, I would choose the multivariate, in the hopes that our reverse causality could get **diluted**.

Model	All features			10 features		
	R^2	MSE	MAE	R^2	MSE	MAE
Multivariate Linear Regression	**63.8%**	**7.600**	**1.570**	57.7%	8.886	1.648
Multivariate Lasso	63.0%	7.774	1.528	57.7%	8.879	1.637
Multivariate Ridge	63.8%	7.600	1.570	57.7%	8.886	1.648
Univariate Linear Regression	**53.0%**	**10.510**	**1.673**			
Univariate Lasso	53.0%	10.510	1.673			
Univariate Ridge	53.0%	10.510	1.673			

Figure 44: Main Results

7 Time Series Analysis

The first problem we faced was to extract time series from the dataset. There is a `features` dataframe in the original dataset, however we quickly found out that we could not use it as there were lots of problems such as series of different lengths and errors in extraction that led to having malformed waveforms, flat on 0 - especially in the first third of the series.

Therefore we chose to extract our own data points from the `small` set of songs: snippets of 30 seconds from 8 thousand tracks, genre-balanced among the **8 main genres**. We extracted using the same sampling rate of the original songs (to get accurate waveforms) and then reduced the many millions datapoints with a first approximation pass, which consisted in resampling the series down with the high-quality resampling algorithm `kaiser_best` from *librosa*.

With this approach we used almost all of the songs (we had to drop only 3) and extracted series that are of the exact same length. We have used the whole 30 seconds of each song snippet while also keeping the amount of data points manageable for performing the tasks required. Moreover, doing one and only one high quality extraction also meant that we spent a lot of time on this extraction itself (~36 hours) and minimized the computational time spent for the tasks, which was desirable, so that for the tasks themselves we could spend more time on e.g. hyperparameter tuning.

Our result is a dataframe of **7997 time series**, made up of **2699 features** each.

7.1 SAX-PAA approximations

One of the crucial tasks for this time series analysis was approximating, and understanding with which parameters. Since we had about 2700 data points for about 8000 songs we realized that we could not run clustering **at all** without having some approximation as the distance calculations were way too massive for our computers. Therefore, we wrote a custom, multiprocessing grid search and applied it in a few ways.

7.1.1 Grid searching on K-means

First, as clustering was the main obstacle with full time series, we run a Grid Search on k-means on the **PAA number of segments** and the **SAX number of symbols**; we tried this with a range of values for the **number of clusters k**. Our results came out that the best clustering results, according to the SSE, were those with the roughest possible approximation (very small number of segments and symbols) and with the biggest possible number of clusters. The following table exemplifies the trend of these results.

Segments	Symbols	K	SSE Euclidean	SSE DTW
5	5	9	0.05	0.03
5	5	8	0.06	0.05
5	5	7	0.09	0.06
5	5	6	0.11	0.08
5	5	5	0.15	0.12
10	5	9	0.16	0.12
5	5	4	0.18	0.15

Figure 45: Grid search on clustering for approximation, top results

We believe these are artificial and misleading. Of course when using the SSE metric, that calculates intra-cluster error, the error will be smallest when the time series are as similar to each other as possible, which is exactly what we are doing when using a very low number of segments and symbols. Moreover, SSE will be smallest when the k number is the biggest in our search range. These results in our opinion have nothing to do with our data themselves but they basically describe how the approximation and the k-means work together. Also, we could not use another metric than SSE (e.g. silhouette score) due to the calculations complexity.

7.1.2 Grid searching on KNN

Therefore we figured it was best to grid search on one of the classification models and apply those values of segments and symbols across the board. So that's exactly what we've done by grid searching with KNN on these parameters:

- number of segments (ranging 5-1000)
- number of symbols (ranging 5-100)
- number of k-neighbors (ranging 3-31)
- distance metric Euclidean or Manhattan.

The total number of **KNN models learned** is **21280** and for simplicity of reporting we used the **accuracy** as for the evaluation metric. These were the top results of our search:

Model Id	Segments	Symbols	N_neighbors	Euclidean accuracy
1381	**130**	**20**	**21**	**0.1725**
1378	130	20	15	0.1713
1401	130	30	3	0.17
1380	130	20	19	0.1694
1351	130	10	17	0.1681
1379	130	20	17	0.1681
1543	130	80	9	0.1675
403	30	50	25	0.1662
1542	130	80	7	0.1656
1405	130	30	13	0.1656

Model Id	Segments	Symbols	N_neighbors	Manhattan accuracy
404	30	50	27	0.1706
3027	280	40	9	0.1688
1401	130	30	3	0.1681
1379	130	20	17	0.1681
1513	130	70	5	0.1656
1405	130	30	13	0.1656
2970	280	20	7	0.165
403	30	50	25	0.165
1384	130	20	27	0.165
2336	205	75	27	0.1644

Figure 46: Grid search on KNN for approximation, top results

The top result with the Euclidean distance (accuracy 0.1725) has segments and symbols **130 and 20**. This combination appears often among these top 20 results (1st, 2nd, 4th, 6th place and more) and appears also in the top accuracy by Manhattan metric (4th and 9th place and more) so we chose these as parameters for every SAX approximation we've done in this third module.

PAA segments: 130; SAX symbols: 20.

7.2 Clustering

For the clustering task we chose to use k-means and run again a grid search on the k hyperparameter. However, we've encountered once again an impossibility to extract silhouette scores as they require computing pairwise distances among all possible combinations of two time series - which would amount to $\binom{7997}{2} = \mathbf{31\ 972\ 006}$ distances for every model computed through the grid search. As for using the SSE to choose the best value of k, the same considerations made above for SAX apply here: using the SSE would only artificially lead to us selecting the biggest k in the range tested. Therefore, we resorted to a low-tech trick: plotting the SSE as it decreases through k increasing and choosing the k where the decrease slows down. Here is that plot for Euclidean and Dynamic Time Warping distances:

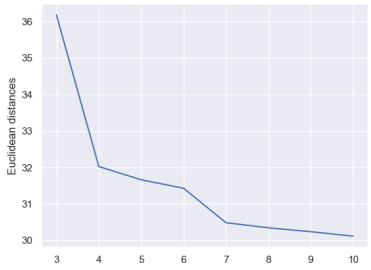


Figure 47: SSE plot with Euclidean dist.

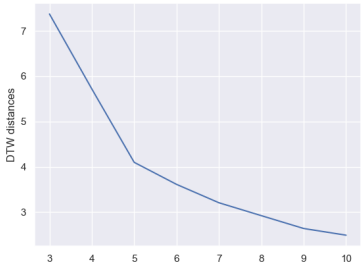


Figure 48: SSE plot with DTW dist.

Moreover we looked at the genres distribution over the various clusters as k decreased and used that to help figure out the best k: we set a k = 10 and decreased that value, stopping when decreasing k more would have led to just one big cluster with 96% of the points. With these methods we chose **k = 5** and the **DTW**. DTW because its SSE is lower but also because it makes a lot of sense to use DTW for our problem which is trying to cluster audio waveforms which might very well have common patterns but in different points of the song.

This is our centroids characterization:

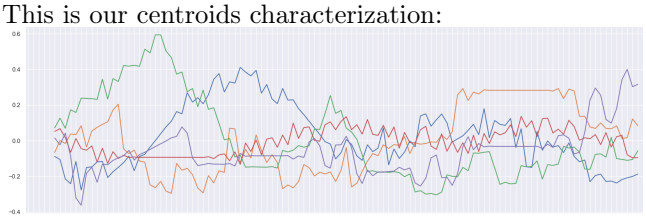


Figure 49: Centroids with 20-moving average smoothing

As this characterization involves audio waveforms, it's not very informative and it makes difficult to say what each cluster represents by looking at this graph. We can point out that 4 out of 5 centroids on two occasions converge in coordinates; these happen both in the first third of the series, while in the remaining space they have a more varying distribution; moreover, we can track the distribution of features such as genre in each centroid, and can show that the 5 centroids actually capture somewhat different proportions of each genre.

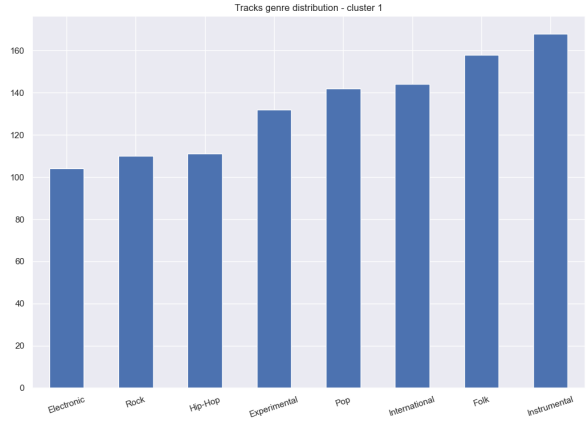


Figure 50: Genre distribution in k-means cluster 1

For instance, the first cluster has a higher proportion of *instrumental* and *folk* music (**"group 1"**) and a lower proportion of *electronic*, *rock* and *hip-hop* (**"group 2"**). These two groups of music are arguably quite different between each other, and very similar among themselves - i.e. instrumental and folk are similar; electronic, rock and hip hop are similar; group 1 and group 2 are however more different. This is, in our opinion, what we would expect to see in a clustering that succeeds in capturing a bit of our *genre* variable.

8 Motif Analysis

For Motif Analysis task we decided to apply a partitional prototype-based clustering algorithm among our time series and take the prototypes as representative time series of our data set.

8.1 Data Preparation

We applied **k-means** and entered the centroids as input of our motif analysis. We used these centroids as representative time series of our data set. As there are 8 unique genres, as a naive approach we decided to use **k=8** in order to try to capture some information about track genres. First of all we scaled our time series using **TimeSeriesScalerMeanVariance** and then we approximated them with **SAX** algorithm, using the number of segments found in the previous section, 130. After that we applied our k-means algorithm with k equal to 8 and **dynamic time warping** as metric.

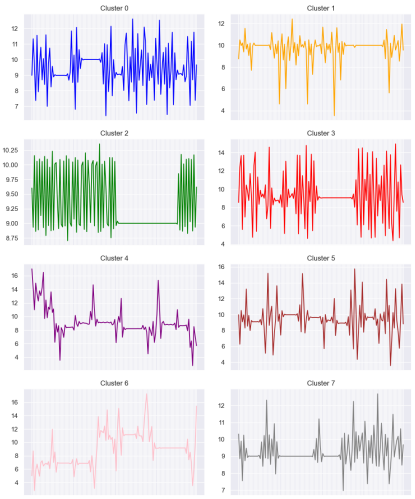
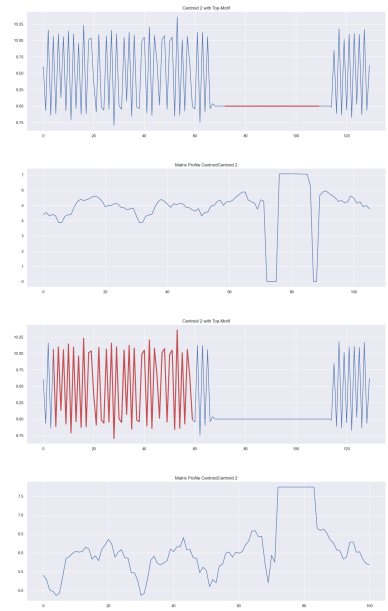


Figure 51: K-Means centroids

8.2 Matrix Profile

After this time series preparation we built a matrix profile for each centroid. **Matrix profile** construction is fundamental for the motif analysis, so we did multiple attempts to find the best subsequence **length** to compute the pairwise distances. We have observed that with small values of our subsequence window most of the straight lines in the centroids were involved and the resulting Matrix Profile was skewed. For this reason we decided to use the smallest window size that was less affected by the nature of our centroids. For these reasons we used 15 as final result, using the same length used for the shapelets in the classification task, not by chance but in order to easily compare them later on. The centroid 2 was the only one to still present some problems, due to its shape, so we continued to increase the length of the window and we reached a size of 30. To better understand the problem we plotted two different Matrix Profiles, the first one with a small value of the window length and the other with our final window length.



Here you can clearly see how the Matrix Profile change radically according the window length used. If we use a small value the straight line is returned as motif but if we slightly increase the window size a new interesting motif appears. For our purpose it is not interesting to return straight line as motif since it is a centroid feature that does not represent time series.

8.3 Motif Discovery

Once we obtained our 8 Matrix Profiles we applied the motif algorithm. As parameter we didn't consider a minimum distance between indices for each sub sequence identified in a single time series and we didn't put a limit of the number of motifs, in order to generalize our task. We put instead a limit on the number of neighbors, since for our last conclusions we will just use the first one discarding the neighbors. Then we started the discovery process: for each motif found, we found neighbors that are within at least the double of motif mp of the first, so we used a radius equal to (2*motif mp). After discovering all the motifs in our 8 time series we ordered them according to their minimum matrix profile value for each motif set and plotted them in a histogram.

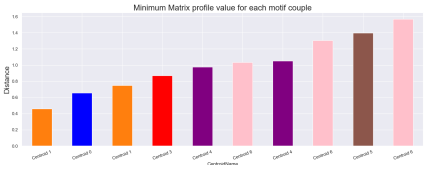


Figure 52: Minimum Matrix profile value

From the histogram above we can see that the first 7 motifs have small values of Minimum Matrix Profile distance; then we have a jump for motifs with distance above 1.2. For this reason we decided to continue our analysis with just motifs with a distance lower than 1.2. We plotted our first 7 motifs in the following image in order of their Minimum Matrix Profile distances for each couple found.

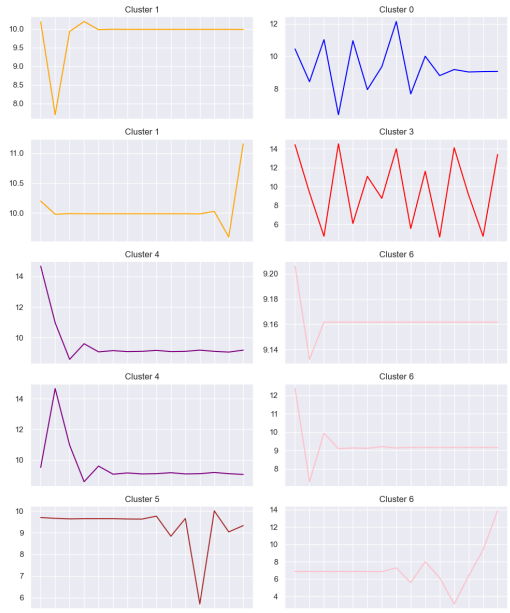


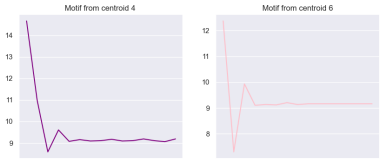
Figure 53: Motifs

As we can see, some centroids showed more motifs and others are discarded because their minimum matrix profile value was too high.

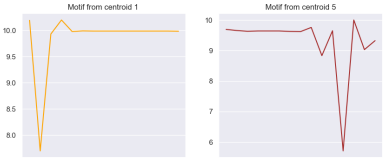
8.4 Motifs Comparison

From the plot in the previous section we have noticed that some patterns occurs in different centroids. In order to continue our analysis of motifs we calculated for each one its dynamic time warping distance with the rest of them. We have found tree interesting couple of motifs that showed a small value of dtw distance.

1. DinamicTimeWarping between motifs found in centroid 4 and centroid 6: **3.001**



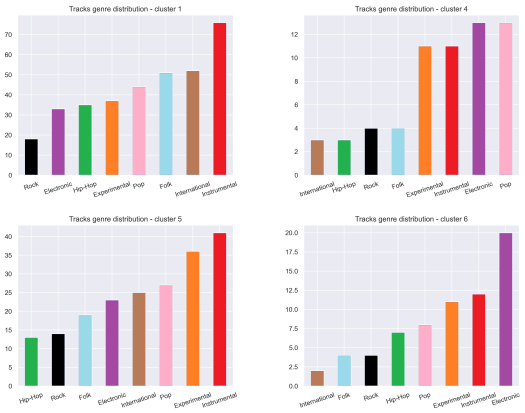
2. DinamicTimeWarping between motifs found in centroid 1 and centroid 5: **3.175**



3. DinamicTimeWarping between motifs found in centroid 1 and centroid 6: **3.493**

8.5 Conclusion

At the end of our analysis we compared multiple feature of time series that were used to compute the centroids in order to find a correlation between features and motifs. The absence of correlation is due to the diversity of times series inside each cluster, since the unsupervised nature of clustering. Observing the genre distribution of our last tree couple found we can observe some similarities. We have plotted the genre distribution of all cluster involved: Cluster 1, Cluster 4, Cluster 5 and Cluster 6.



- Cluster 4 and Cluster 6: Both cluster share the fact that the most frequent genre inside their cluster is electronic.
- Cluster 1 and Cluster 5: Both cluster share the fact that the most frequent genre inside their cluster is instrumental.
- Cluster 1 and Cluster 6: For our last couple that showed the lower dtw distance we can observe that for cluster 1 the most frequent genre is instrumental while in cluster 6 instrumental is in second place.

This reasoning is too simplistic and this genre similarity distribution is probably due by chance but we decided to show it for our starting idea.

9 Classification Task

For this part of the project we decide to use two different classification datasets and analyze them with shapelets and without, with all complete Time Series. So the two datasets for classification task are:

- Full Dataset
- Approximated Dataset with SAX

The classification tasks that we decide to do do is a **Multi-Class Classification on Generes**. That's because we used `fma-small.zip` which is perfectly balanced into 8 genres and we think it's very interesting to verify if using Time Series we can recognize them. First of all we scaled our Time Series using *TimeSeriesScalerMeanVariance* in all the cases. Furthermore we have to keep in mind that for this classification the random classifier threshold to try and

exceed is 12%. Of course we encoded the genres, and below is shown how.

Genre	Encode as
Electronic	0
Experimental	1
Folk	2
Hip-Hop	3
Instrumental	4
International	5
Pop	6
Rock	7

Figure 54: Genre Enconding

9.1 Shapelet Retrieval

Since we decided to use two types of dataset(with approximated and complete time series), we have to retrieve from each one their two sets of shapelets. Starting from the **not approximated** dataset we decides to use the *"grabocka-params-to-shapelet-size-dict"* and they return us the perfect match of shapelets for our dataset. It recommend to use, for 7997 Time Series of songs, 2699 points long, just **8 shapelets, 251 long**.This is a really low number of shapelets so we decided to increase it by 3 times, therefore using **24 shapelets, long 250** (for evenness). Below is shown the dataset in 1-D dimensional contiguous flattened array with example locations of shapelets.

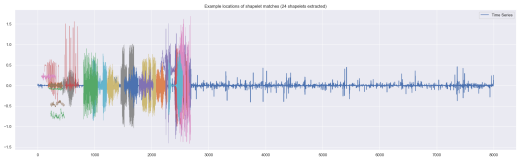


Figure 55: dataset and shapelets

Now we can talk about the **approximated** version of the dataset .So we have 7997 Time Series but they are length 130 points. Even here we use the grabocka methods and he suggest to use 6 shapelets with length 15, but also in this case we decide to increase the number of shapelets to **24 with length 15**. Below there the same graph with 1-D dimensional array with example of shapelets.

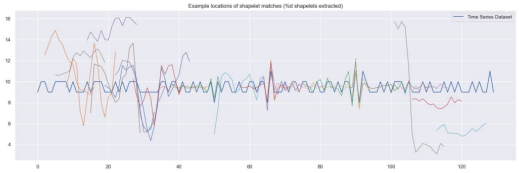


Figure 56: dataset approximated with Sax and shapelets

We can compare these shapelets with the motifs retrieved in section 8.3.

9.2 Similarities and Difference between Shapelet and Motif

The comparison between shapelets and motifs was made calculating the *Dynamic Time Warping distance* between their points. In the histogram Figure 57 are shown the minimum distance found using this metric distance. We decided to plot only distances minor than 2.5 because we saw that most are in this range.

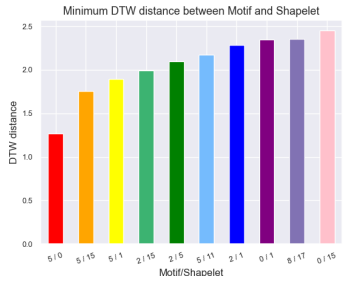


Figure 57: Distance between Shapelet and Motif

After that we decided to plot **the best 10 comparisons** and draw our conclusions from those.

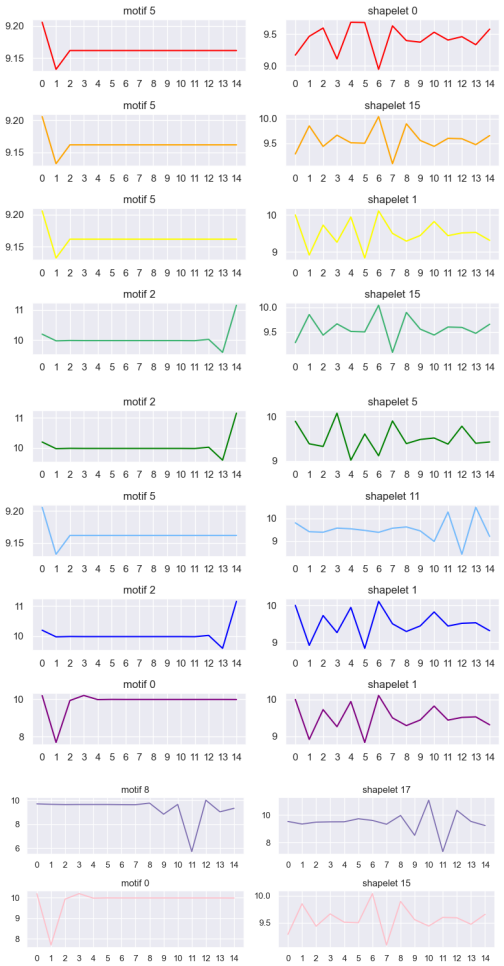


Figure 58: Best 10 Shapelet and Motif

We can notice that all comparisons with motif that have straight line are retrieved (that are not included in the shapelets) and similar peaks that we can see if we just look at the single motif or shapelets. For completeness we decided to plot the other **18 shapelets** retrieved too.

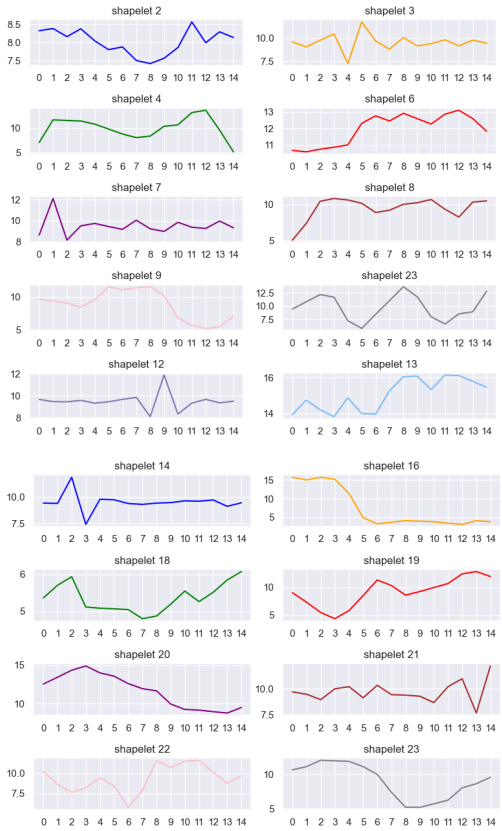


Figure 59: Shapelet

9.3 Classification with the Complete Dataset

Here follows a multi-class classification over 8 genres. We decided to use two different classifiers that operate in a very different way, **KNN and Random Forest**. This because we can see that in the first part Random Forest has very good results and with KNN we can use DTW as a metric (this we used only in the approximated dataset because we don't have enough computational power).

9.3.1 KNN study

Starting from KNN we performed **KNN with shapelets and without**. In both we Grid searched to tune the best hyper-parameters.
So for **KNN using shapelets** classification we have **n-neighbors=17**, **weights based on distance**, and **euclidean** as a **distance metric**. We have a very good **accuracy** of **20%** on all the classes and an F1 score - which is very relevant for our task - averaging 0.2. This classification is one of the best that we have and the class best recognized is *Rock* with an **F1=0.24**.
Instead for the **KNN not using shapelets** classification (we do KNN on all the Time Series) the tuning of hyper-parameters is only different in regards to the **number of n-neighbors** which is **29** and the overall results are worse. The **accuracy** is just **18%**, the average F1 0.18 and the best class becomes *Hip-Hop* with an **F1 of 0.3** but strangely *Rock* became the worst class with an F1 of 0.1. In the Figure 60 are the results.

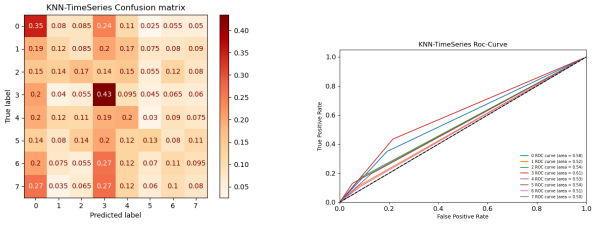


Figure 60: KNN with TS,Confusion Matrix and ROC

This probably happened because using shapelets with KNN we can better recognize classes that are not so well defined and the results are more homogeneous: the shapelets better classify the single class because of the curse of dimensionality, we pass from 2799 features to 250 and this helps KNN. For that reason we can conclude that, for KNN, shapelets help a lot the classification and are worth using even if they take precious CPU time. All the results are in Table 65.

9.3.2 Random Forest study

After KNN we performed **Random Forest** classifier on the complete dataset. In this case we always perform the tuning of the Hyper-Parameter and we have that for all the classification the **best criterion is gini**, we use **max log2 features** and the best **number of estimator remains 100**.
For the **shapelets Random Forest** we decide to use (after the Random Search) a **max-depth=14,min-samples-split=3 and min-samples-leaf=50**.Here we have the best results of all our classification, with an **accuracy of 23%** and an average F1 of 0.22. We can say that using shapelets and Random Forest is a very good choice for our task. The best class like in in KNN is *Rock* and the worst is *Experimental* that it is almost not recognized. In Figure 61 we can see all the results.

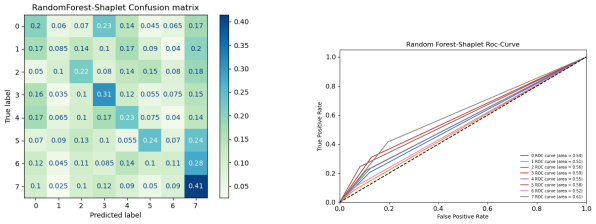


Figure 61: RF with shapelets,Confusion Matrix and ROC

But there is something strange, in fact if we look at the **Random Forest upon all the Time Series** we have the hyper-parameters that change in **max-depth=13,min-samples-split=5, min-samples-leaf=1** but the interesting things is that we have a **worst accuracy(upon 18%)** the avg F1 is 0.18, but looking at the single class we can see that recognizes them better, because the worst class, which is *Electronic* in this case,have a F1 equal to 0.14, that is much better than the 0.1 of the *Experimental* ones using shapelets.
So we can conclude that the **Random Forest using shapelets is one of the best model** even if it is slower than the other one.But we have to be careful if we want to better recognize class or if we are

interesting in looking at the full classification evaluation. Because we could have good rates in general, but the evaluation of single class are to much disperse.In the Table 65 you can observe all the results.

9.4 Classification with approximated dataset with SAX

The approximated dataset with **Sax** is composed of **7997 Time Series** but its **length** changes from 2699 to **130**. This is a very big approximation but we hope that classification works well.

9.4.1 KNN study

First of all KNN. Like we do for the complete dataset, we perform a classification using **shapelets** that in this case are always **24 but their length is 15 points**, and a classification on the complete Time Series. But now we can use not only Euclidean and Manhattan distance, but **Dynamic Time Warping** too. So of course we tested it in shapelets and upon all the Time Series.
Starting from the **shapelet model with KNN** we perform a Grid Search on the hyper-parameters and now we have a number of **neighbors of 19** and it's better to use the **Manhattan** distance instead of Euclidean like we do in the complete dataset. Now what is interesting is that we have a very **low accuracy, 16%,** compared with the complete dataset and if we look at the single class the average F1 decreases to 0.16 and more than one class are not recognized well like we see for *Pop* genre. This is quite interesting because if we analyze the classification **with the Time Series and KNN** we have increased results! That is exactly the opposite situation of the complete dataset! Here we have the opposite situation about **the curse of dimensionality** and this came from the fact that we do a **double feature reduction**, one using Sax and the second one with shapelets. We have to consider **21 neighbors** instead of 19 and the other metrics are the same. But the juicy thing is that the **accuracy increases to 17%**, the avg F1 remains the same at 0.16 but the **average of the Precision and the Recall increase**. This means that the single classes are better recognized without using shapelets and if we look at the single class, the worst has an F1 of 0.12 which signifies that it's recognized, while with the shapelets the worst class was hardly seen.
Below are the Confusion Matrix and the Roc of the KNN with the Time Series Classification.

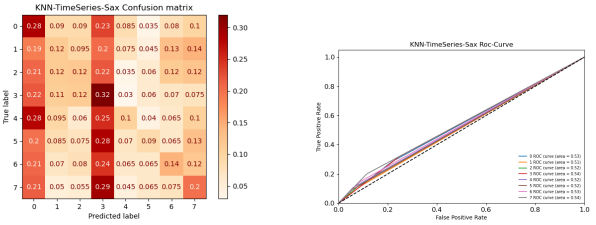


Figure 62: KNN with TS,Confusion Matrix and ROC
Analyzing these **two classification** (with shapelets and with Time Series) **using DTW** we can only confirm what we observe before. Changing the distance metric **has accentuated the difference** even more than using the Manhattan ones. We can observe that the accuracy remains the same if we use shapelets, and all the value are practically the same but the class change. So we can say that in **shapelet classification the distance metric does not effect to much the results.**
On the contrary we can observe that if we use the complete **Time Series classification the results increase** a little bit. The accuracy switch to 18% and all the other metrics improve their results. Confirming that the Time Series classification without shapelets is the best choice if we decide to use Approximation with SAX. And it's faster too!
Below we can observe the Confusion Matrix and the Roc curve using DTW upon all the Time Series classification and in the Table 65 there are all the results.

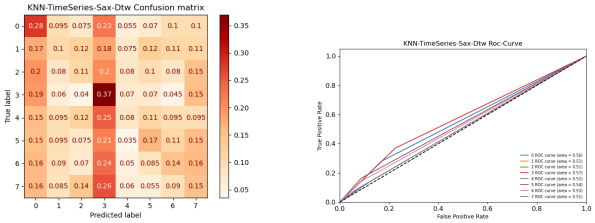


Figure 63: KNN with TS-DTW,Confusion Matrix and ROC

9.4.2 Random Forest study

Last but not least, Random Forest. We like before perform a Random Search upon the hyper-parameter both for shapelet classification that for Time Series classification. The **criterion is always Gini** but for the **Random Forest using shapelets** we have a **max depth of 25, min samples lift of 30 an a min samples leaf of 2**. The **accuracy** of this methods is of **17%** and the other metrics has an avg of 17% so we can see that is not one of the best, but not even one of the worst. It interesting to notice that the worst class is not recognized well and is the same of the KNN classification with shapelets(*Pop*).
Below we can observe the Confusion Matrix and the Roc curve of this methods.

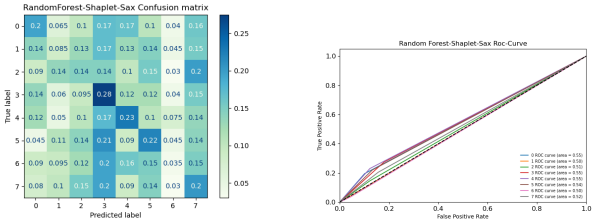


Figure 64: RF with shapelets,Confusion Matrix and ROC
Instead looking at the **Time Series classification**, the hyper parameters change, we have **none max depth, min samples lift of 10 and a min samples leaf of 10**. The results are quite similar to the previous one but the single class are recognized better and we can see that all the class exceed the 12% threshold.

9.5 Classification: conclusions

So what did we discover at the end of this long classification task? It all depends on the case. We can observe that if we approximate or not the dataset all the considerations change. And this is quite important for our classification task. If we have to choose which one of the datasets is better we can say of course that the **complete** is the choice and in particular, **Random Forest with shapelets**, however we can confirm that the time spent in computation is much more instead of using the **approximated dataset**, obviously.
However with the "small" dataset, Random Forest with shapelets is not the best choice. With the Euclidean distance, using the **Time Series classification with KNN** gives us good results in a very small time. The **DTW distance**, however, is not worth using, because it takes a very long time (much more than the Random Forest with shapelets), **and** the results are worse.
In the following table we made the distinction of the best class in our test performance metrics, the worst class and the average across all 8 classes.

Sax approximated						Parameters	Model	Parameters	Not approximated				
	Class	Accuracy	F1	Precision	Recall				Class	Accuracy	F1	Precision	Recall
avg		0.17	0.16	0.17	0.17	criterion="gini", max_depth=None, min_samples_split=10, min_samples_leaf=10,	**Random Forest**	criterion="gini", max_depth=13, min_samples_split=5, min_samples_leaf=1		0.18	0.18	0.18	0.18
best class	Instrumental		0.24	0.21	0.28				Hip-Hop		0.22	0.22	0.23
worst class	Experimental		0.12	0.15	0.1				Electronic		0.14	0.15	0.14
avg		0.17	0.17	0.17	0.17	criterion="gini", max_depth=25, min_samples_split=2, min_samples_leaf=30,	**Random Forest with shaplet**	criterion="gini", max_depth=14, min_samples_split=3, min_samples_leaf=50		0.23	0.22	0.22	0.23
best class	Instrumental		0.22	0.21	0.23				Rock		0.3	0.23	0.41
worst class	Pop		0.05	0.1	0.04				Experimental		0.11	0.17	0.09
avg		0.17	0.16	0.18	0.17	n_neighbors=21, weights="distance", p=1	**KNN**	n_neighbors=29, weights="distance", p=2		0.18	0.18	0.19	0.2
best class	Experimental		0.21	0.16	0.32				Hip-Hop		0.3	0.22	0.43
worst class	Hip-Hop		0.12	0.2	0.09				Rock		0.1	0.08	0.12
avg		0.16	0.16	0.16	0.16	n_neighbors=19, weights="distance", p=1	**KNN with shaplet**	n_neighbors=17, weights="distance", p=2		0.2	0.2	0.2	0.2
best class	Instrumental		0.2	0.2	0.2				Rock		0.24	0.22	0.27
worst class	Pop		0.09	0.1	0.08				Experimental		0.13	0.14	0.12
avg		0.18	0.17	0.17	0.18	n_neighbors=19, weights="distance", metric="dtw_sakoechiba"	**KNN & DTW**						
best class	Hip-Hop		0.25	0.19	0.37								
worst class	Instrumental		0.11	0.16	0.08								
avg		0.16	0.16	0.16	0.16	n_neighbors=21, weights="distance", metric="dtw_sakoechiba"	**KNN & DTW with shaplet**						
best class	Electronic		0.2	0.2	0.2								
worst class	Experimental		0.11	0.12	0.1								

Figure 65: Main Results, Time Series Classification

10 Sequential Pattern Mining

We started sequential pattern mining on our time series with the usual SAX approximation (130 segments - 20 symbols). Sequential pattern mining with the library we used (*prefixspan*) is, however, incredibly computational intensive. The parameters we have are the **m**, length of the pattern we are finding, and the **minimum support** to have for each pattern extracted.
As we increase the m, the complexity increases steeply; the maximum we were able to compute was **m=10**. With this length, we found support as high as **7758** out of 7997. These are the top-support patterns found of length 10 - as you can see, very similar to each other:

- (7758, [9, 9, 9, 9, 9, 9, 9, 9, 9, 9]),
- (7757, [10, 9, 10, 10, 10, 9, 9, 9, 9, 9]),
- (7757, [9, 9, 9, 9, 9, 9, 10, 9, 10, 10]),
- (7755, [9, 10, 10, 10, 9, 9, 9, 9, 9, 9]),
- (7755, [9, 9, 10, 9, 9, 9, 9, 9, 9, 9]),
- (7754, [9, 10, 10, 9, 9, 9, 9, 9, 9, 9]),
- (7754, [9, 9, 9, 10, 9, 10, 10, 9, 9, 9]),
- (7754, [9, 9, 9, 9, 9, 10, 9, 10, 10, 10]),
- (7753, [9, 10, 9, 10, 10, 9, 9, 9, 9, 9]),
- (7753, [9, 10, 9, 9, 9, 9, 9, 9, 9, 9]),
- (7753, [9, 9, 10, 10, 10, 9, 9, 9, 9, 9]),
- (7753, [9, 9, 10, 10, 9, 9, 9, 9, 9, 9]),
- (7753, [9, 9, 9, 10, 9, 9, 9, 9, 9, 9]),
- (7753, [9, 9, 9, 9, 9, 9, 9, 10, 10, 10])

Considering that the number of time series on which we're running is **7997**, the support values found are very high. This is mostly because after SAX approximation our time series look very similar, and this similarity is compounded by the sequential pattern mining, which only considers the progressive order of the symbols and not any symbols caught in between. Therefore, considering the enormous support (around **96%** of the dataset), the only way to extract information from sequential pattern mining is to interpret the specific patterns found. This, however, is also not feasible because we're analyzing patterns of symbols extracted from audio wave forms. It does not tell us much that the most frequent pattern is [9,9,9,9,9,9,9,9,9,9]; 9 is close to the median symbol (out of 20) and it's understandable that it would be very frequent in a waveform. Also of note is that the only symbols that appear are **9** and **10**, close to the medians of the wave forms. So, studying the patterns themselves is not feasible in this kind of audio data. Something else of note is that, given the high similarity of these patterns, it is very probable that 2 or more of them appear contiguously in the same time series. For instance, it is possible that just before our last pattern in the list above, [9,9,9,9,9,9,9,9,10,10,10], appears the first pattern, [9,9,9,9,9,9,9,9,9,9], since the latter ends with seven 9s and the former starts with seven 9s. There are many of these "patterns" to be found in our patterns. This makes us suspicious of there being longer frequent patterns to be potentially mined, but this for us has to remain just a suspicion because with the power we have access to we can't increase the value of **m** and verify.

While working on this, we also figured out another way to do **"sequence-mining"**, which we discuss in the last chapter as an additional part to this study.

11 Advanced Clustering

For this task, at the beginning we decided to use the original dataset with 42 columns but cutting some of the features that are not interesting to analyze, so we have a new dataset with 12 columns:

- (album, comments)
- (album, date created)
- (album, favorites)
- (album, listens)
- (artist, comments)
- (artist, date created)
- (artist, favorites)
- (track, date created)
- (track, duration)
- (track, favorites)
- (track, interest)
- (track, listens)

All of these columns are numerical and we normalized them with StandardScaler. So our dataset is composed by **98247 records and 12 columns.**

11.1 X-Means

We chose **X-Means** for advanced clustering and what we do at the start is selecting **2 initial centers**. After that we do clustering setting **3 different types of Kmax** running 3 different clustering and after that we do a comparison between the 3 SSE for choosing the best number of max centers that we could have for our dataset.
The first **Kmax** that we prove was equal to **5**. So we have 5 centroids and centers and we decide to plot 2 combination of our columns for better visualize them, so we plot *(track, listens)* and *(track, date-created)* in the first and *(artist, favorites)* and *(album, favorites)* in the second one for all the Kmax. We can observe that the cluster for the first combination of features are very well separated, we can see 4 cluster very well so for the orange one,this starts from the track listens = 4 and cover all the dates, the blue one cover the date down to value 0.0(due to normalization) and the green one to from 0.0 to 1.5 for the date , all that in a range of listen 1-2.5.Instead the red one that is also the only one identifiable in the other graph cover all the small value of listens on all dates. The centroids are all in the low part of track listens, and just one is in the upper part, instead in the second ones they are all in a range of low value of album and artist favourites.

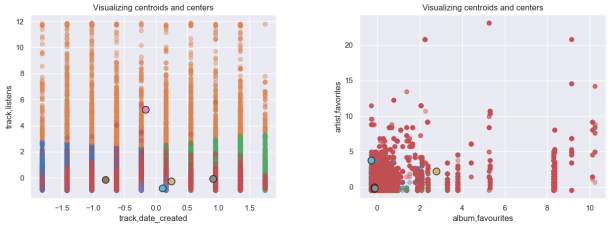


Figure 66: Visual Representation with 5 centroids
The second one was **Kmax equal to 10** and in the Figure 67 are showed the graphical representations.

Here too the first graph is well representative of the division of clusters and we can see 7 cluster very well that describe the division of the value for this two variables. The centroids also in this case are in the low part of listens but now we have 2 in the upper part that divide medium from higher listens. Instead, in the second graph we can identify just 2 cluster that divide the album favourites in low favourite so up to value 4 and highest number of favorites, from 4 to 11. Here the centroids are always all towards the left side but there is one that identifies the height value of album favourites.

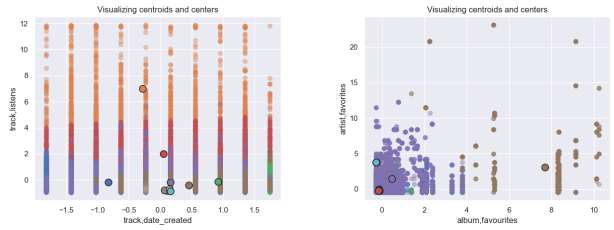


Figure 67: Visual Representation with 10 centroids

And the third one is with **Kmax = 20.** Now we can observe a very good division in all the two graph with different value for each cluster, but of course we can't see all the cluster. Here we have a lot of centroids mostly in the lower part of the graph and the same division for listens medium and height that we recognize before, and in the second part we have a lot of centroids all towards the left side but we can identify also two on the right and one at the top of the chart to identify higher values of favorites.

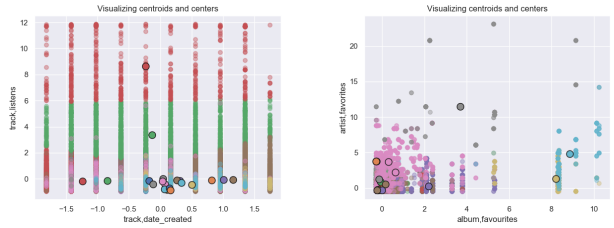


Figure 68: Visual Representation with 20 centroids

At the end we analyze the SSE of the 3 clustering for choosing the best one:

	Kmax = 5	Kmax = 10	Kmax = 20
SSE	550743.8	457048.4	266610.8

Figure 69: SSE results

So in conclusion we decide to chose the **Kmax = 10** because we can have a good value of SSE even if it's not the best, we can better recognize all the most important cluster that better divide our dataset. Of course Kmax = 5 is not enough and Kmax = 20 is too specialized since it recognizes clusters that could be redundant or too small and which could involve values that are too similar in different clusters.

12 Transactional Clustering

Talking about Transactional Clustering we decide to use the same dataset of the Advanced ones but transforming all the numerical columns into categorical by binning them.

12.1 K-Modes

We decided to to do **Transactional Clustering with K-Modes** and we tested with SSE different number of clusters. So we decide to talk about **3 and 6 clusters** because they are very interesting.

For the **K-Modes** setting **3 clusters** we have that the labels are not so well separated and the cluster are very similar. The **SSE** for this Clustering is **632922.0**. Below are showed the table with the details about the centers of each centroid.

K-Modes with 3 Cluster	N.of record	Centers
Cluster 1	39914	('no-comments' '2016' 'no-favorites' 'medium-listened' 'no-comments' '2016' 'low-favorites' '2016' '180' 'low-favorites' '1501' 'medium-listened')
Cluster 2	26758	('no-comments' '2009' 'no-favorites' 'low-listened' 'no-comments' '2009' 'low-favorites' '2009' '180' 'low-favorites' '546' 'low-listened')
Cluster 3	31647	('no-comments' '2012' 'no-favorites' 'low-listened' 'no-comments' '2012' 'low-favorites' '2012' '60' 'no-favorites' '302' 'low-listened')

Figure 70: Centroids and relative centers

We can observe that the clusters are not so different, the number of records is quite balanced into the 3 clusters and the centers are similar regarding specific variables and different regarding others. For example looking at the commented or not album or artist, in all of them there are no comments. Instead all the date variables are different between the clusters. The cluster one has not all the values categorized as low as we can observe from the other cluster, but have medium listened album and tracks, so is the most different from the other.

For the K-Modes setting **6 clusters** we have that the label are well separated and the cluster are not very similar so we can identify better all the difference between cluster. The **SSE** for this clustering is **561350.0**. Below is shown a table with details about the centers of each centroid.

K-Modes with 6 Cluster	N.of record	Center
Cluster 1	18543	('no-comments' '2015' 'no-favorites' 'medium-listened' 'no-comments' '2015' 'low-favorites' '2015' '15' 'low-favorites' '1435' 'medium-listened')
Cluster 2	14755	('no-comments' '2011' 'no-favorites' 'medium-listened' 'commented' '2011' 'medium-favorites' '2011' '180' 'no-favorites' '1287' 'medium-listened')
Cluster 3	15719	('no-comments' '2012' 'no-favorites' 'low-listened' 'no-comments' '2012' 'low-favorites' '2012' '146' 'low-favorites' '302' 'low-listened')
Cluster 4	13109	('no-comments' '2016' 'low-favorites' 'medium-listened' 'no-comments' '2016' 'low-favorites' '2016' '180' 'no-favorites' '56' 'medium-listened')
Cluster 5	21751	('no-comments' '2009' 'no-favorites' 'low-listened' 'no-comments' '2009' 'low-favorites' '2009' '8' 'no-favorites' '342' 'low-listened')
Cluster 6	14442	('no-comments' '2010' 'low-favorites' 'low-listened' 'no-comments' '2010' 'low-favorites' '2010' '60' 'low-favorites' '266' 'low-listened')

Figure 71: Centroids and relative centers

Changing the number of clusters from 3 to 6 the distribution of data in clusters is more sparse and we can better divide the number of records. Looking at the centers, they are quite different, for the date variables are different between the clusters and the track duration and interest are different too. On the other hand the *listened* and *favourites* (regarding album, artist and tracks) are generally low or medium and combinations of these occur in clusters in different ways. Generally there are no album comments or artist comments, but in cluster 2 artist has comments, and this is what characterizes this cluster.
So in conclusion we can say that for this task we prefer to use **6 cluster** instead of 3 because of the distribution of the data, the **better SSE** and the more **differentiated centroids**.

13 Explainability

For this last task we try to explain one of the black boxes used in Module 2 Advanced Clustering. At this point we just have to choose which classification is more interesting to analyze, starting from the fact that we always use a dataset called *complete* that could have different number of features and after that classification we use a dataset with 10 specific variables called *small*, we decide to select one of the classification with the *small* version. This because we observe that the results are very good compared with the "complete" ones, furthermore we think is very interesting to look at what happened in a subsection defined by us.
Which one did we decide for? Observing Table 42 we notice that for Ensemble Classifier, specifically in simple Random Forest (that we selected as one of the best classifiers for us) we have a big difference between the accuracy and the F1 of the "complete" dataset and the "small" one. So we decided to explain that one: **Random Forest with 10 features**.

13.1 LIME-Local Interpretable Model-agnostic Explanations

The **lime** library requires categorical variables to be encoded in a numerical format and the *small* dataset is perfect for this task, since they already are all numerical. Using this method we can analyze individually each one of the values that our target variable *(album,type)* could have. So what we do is to select and **analyze 4 instances** of our dataset that return each **unique album type values** and understood **why this happens**,which of the 10 variables are included and what is the top selected ones that are binding in making a decision.

13.1.1 Album

For an instance classified as *Album* the explanation points out how not having any of album engineer, artist bio, album producer and artist website is decisive. And this is observable even looking at the "NOT" side of the other features.

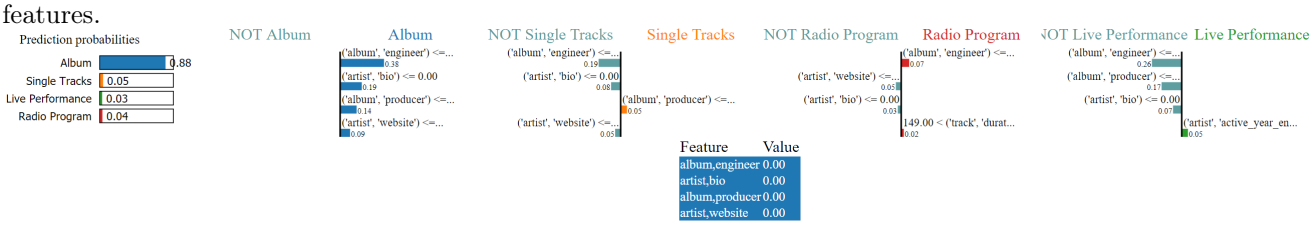


Figure 72: Explain Album

13.1.2 Live Performance

If the instance is classified as *Live Performance* we have, on the contrary of *Album*, that an album that has an engineer and a producer, an artist that has a bio and a track interest <= 500 that in this specific case signify 313.000 as a value. Looking at the other class we can see that this specific instance could be classified as *Single Tracks* for the 30%. This happens because both the two value of the class (album, type) has as the most relevant class album engineer with the same value. But as we can see for the 66% this class is classified as *Live Performance*.

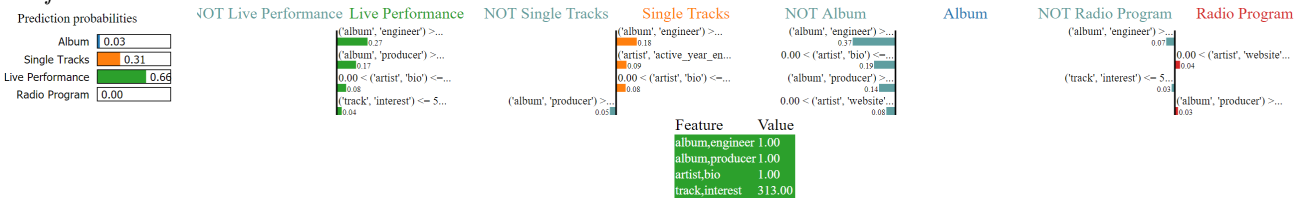


Figure 73: Explain Live Performance

13.1.3 Radio Program

This instance is one of the most complicated to find in the dataset because it is very unbalanced. But at the end we found one; what we can see is that if we have an album engineer we don't have a Radio Program.But if we have an artist website, an album producer and the artist as a bio this classify the instance as *Radio Program*. Another interesting thing that we can observe comparing Radio Program with Album, that is the other relevant prediction probabilities (28%), is that have an album engineer classify the instance as not album like we have in not radio program, hence the fact that if we have an album engineer we have instance that are classified as Live Performance or Single Tracks.

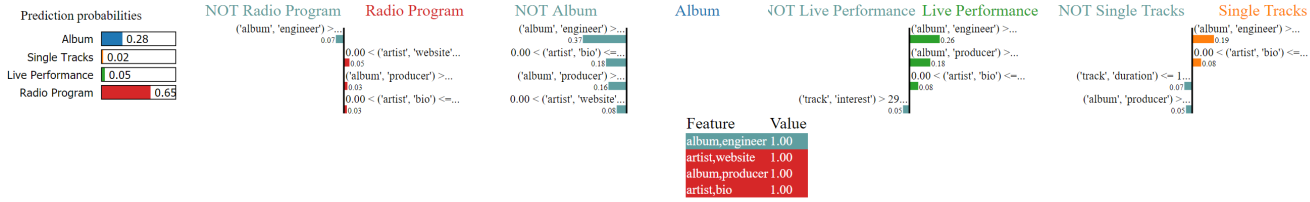


Figure 74: Explain Radio Program

13.1.4 Single Tracks

At the end we have the value of album type *Single Tracks*. To be classified in this way, like we said before we have an album engineer, plus it is necessary that we have a track duration >3 that in our category signify being over 120 seconds, like we can observe in feature value, and another two variables that implies at contrary not be a single tracks are don't have an artist bio and a website. For the artist bio we can say that don't have it signify also don't be a Live performance, and for the artist website signify don't have it imply don't be a radio program.

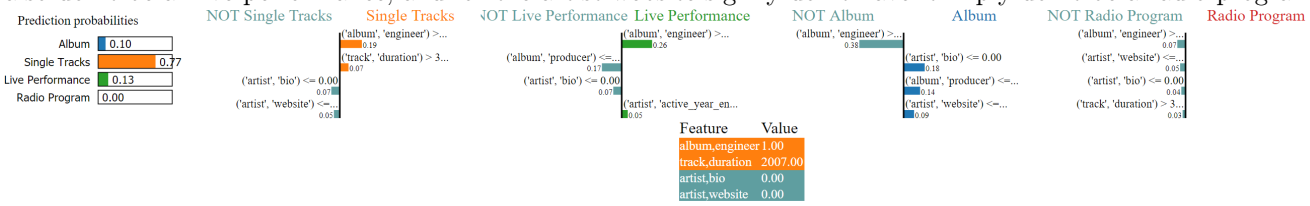


Figure 75: Explain Single Tracks

14 Appendix: Frequent sequence mining

We want to check, among our set of frequent items extracted with **sequential pattern mining**, which of them appear as one contiguous block and with what support; so, we are doing what we're calling **frequent sequence mining** starting from candidate sequences which are the top patterns extracted with sequential mining. These, the first 5 patterns that showed the highest support in the pattern mining section, are:

1. [9, 9, 9, 9, 9, 9, 9, 9, 9, 9] occurs 406 times (as a sequence),
2. [10, 9, 10, 10, 10, 9, 9, 9, 9, 9] occurs 125 times,
3. [9, 9, 9, 9, 9, 9, 10, 9, 10, 10] occurs 135 times,
4. [9, 10, 10, 10, 9, 9, 9, 9, 9, 9] occurs 108 times,
5. [9, 9, 10, 9, 9, 9, 9, 9, 9, 9] occurs 341 times.

There are some interesting results: we can see that the support obtained in the sequential pattern mining (chap. 10, basically the same support for all 5 patterns) doesn't necessarily follow the same distribution of occurrence of that pattern in this sequence mining (shown in the list just above, with highly variable supports). The first sequence occurs 406 times in our set of time series and this represents the 5% of the total number of time series. We can say that this sequence of length 10 appears frequently.
In order to try and give some characterization to this task we observed the *track genre* distributions for the time series that contain one of the 5 sequences in our list. In particular we noticed that there might be some correlations between the 1th and 5th sequences. These two sequences showed interesting features as they both have a similar **distribution of genres** and similar **support**. We plotted their genre distributions in the following histograms:

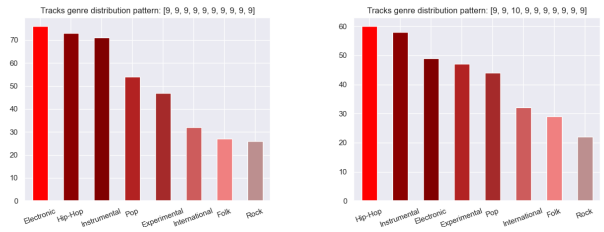


Figure 76: Genre distribution for Time Series that contain 1th and 5th sequences
As specified in chapter 10 we couldn't increase the pattern length, so the fact that 1st and 5th show similar distributions suggests to us that they could be both characterizing a particular distribution and therefore be part of the same, larger, pattern (longer than 10, therefore not discovered with sequential pattern mining).
We also noticed from the genre distribution that if the symbol "10" appear in the sequence extracted, the number of times series of genre Electronic decreases. That's shown in the following histograms of the 1th and 2th sequences:

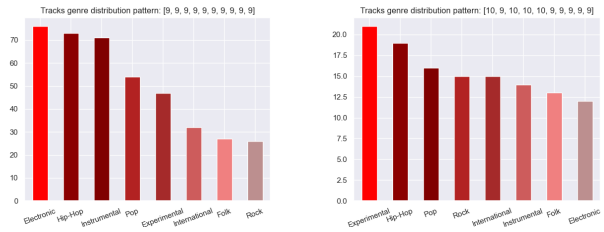


Figure 77: Genre distribution for Time Series that contain 1th and 2th sequences
This can be observed also from the previous comparison. From these two observations it looks like the frequency of genre *electronic* and the frequency of symbol 10 are inversely proportional.

✍

Aknowledgments:
Fabio: I would like to thank my two teammates, it was an absolute joy and super fun to work with them.
Marianna: Thank you for the time spent in reading our work. We wish you enjoy it.
Saverio: It was a fun challenge, I hope you liked our hard work!