Live or Studio Recording Detection

Data Mining 2 2020/21

Group 20

Alexandra Bradan (530887)[1], Alice Graziani (544557)[2], and Eleonora Cocciu (518746)[3]

Contacts: [1]a.bradan@studenti.unipi.it, [2]a.graziani5@studenti.unipi.it,
[3]e.cocciu@studenti.unipi.it

July 22, 2021

Contents

1 Module 1

1.1 Introduction

Music Information Retrieval (MIR) is an interdisciplinary field dealing with the analysis of musical content by combining aspects from signal processing, machine learning and music theory. MIR enables computer algorithms to understand and process musical data in an intelligent way for solving problems like browsing, searching, and organizing large music collections. In this work we focused on **song recording recognition**, that is distinguishing between a studio recorded track and a live recorded performance/radio program. For our goal, we used melodic, structural characteristics as well audience rating, provided by the FMA Dataset, a musical storage, which get access to Commons-licensed audio for 106.574 tracks, 16.341 artists and 14.854 albums, arranged in a hierarchical taxonomy of 161 genres, reunited under 16 parent ones.

1.2 Data Understanding and Preparation

The FMA dataset is composed by four sub-datasets: *Tracks*, *Features*, *Genres* and *Echonest*. Since only 12.32% of total tracks are present in the *Echonest* dataset, in the present report, we focused our attention upon *Tracks* and *Features* datasets, deriving our own version, called *Final Dataset*, as shown in Table 1.

	Tracks	**Features**	**Genres**	**Echonest**	**Final dataset**
Rows	106.574	106.571	163	13.129	103.708
Columns	52	518	5	249	44

Table 1: Datasets shapes.

1.2.1 Data understanding

Since we wanted to predict whether a tracks is recorded or it was live performed, our focus relied upon the **album_type** variable. Originally, the *Tracks* dataset was made up by five *album_type*: *Album*, *Radio Program*, *Live Performance*, *Single Tracks* and *Contest*. We removed tracks labeled as *Single Tracks* and *Contest*, due to less than 1% of records showing these characteristics. By plotting our selected features' distributions (in the following sections we will go more in depth, explaining how we derived them), we noticed that *Radio Program* and *Live Performance* were quite similar among them. As a matter of fact, they are less appreciated by the audience (lower number of tracks' listens, albums' and artists' favorites), with high regard due to the fact they are related to niches. They share the most common creation year (2009), the creation seasons (Winter) and Rock genre presence, too. Instead, they differ in tracks' bit_rate (Radio Programs tend to have less even bit-rate distributions, anyway in the same bit-rate range) and duration (Radio Programs usually have shorter lengths). *Album* are rather recognizable by higher bit-rate, medium duration, releasing in more recent years, preferably during the winter time and through the predominance by the Experimental genre. From what we just said, we got the idea that we can sum up together *Live Performance* and *Radio Program* recording types, getting the new **Live Recording** class label, instead. For clarity reasons, we renamed the remaining album type as **Studio Recording**, thus performing a binary classification. From Figure 1 we can notice a comparison between our target class labels and our selected features: only a slight higher scale factor can be detected for Studio Recording among *Features* dataset's variables, while for the *Tracks* dataset are valid the remarks previously made.

1.2.2 Duplicates

Regarding the presence of duplicate records, we didn't detected directly duplicates rows. Still, we notice a quite interesting behavior: 6.24% artists have at least one duplicate track title, while 1.53% of albums have at least one duplicate song name. Since these duplicate tracks differ only for track's metadata, we didn't considered them to be real duplicates. Instead, we can hypotize that matching title songs having the same duration are part of different versions due to featuring with other artists, since track's base remains the same. On the other part, matching title songs changing their duration are part of track's base contraction/extension and duplicate songs changing bit-rate/genre_top are part of track's base remix/bmp modification.

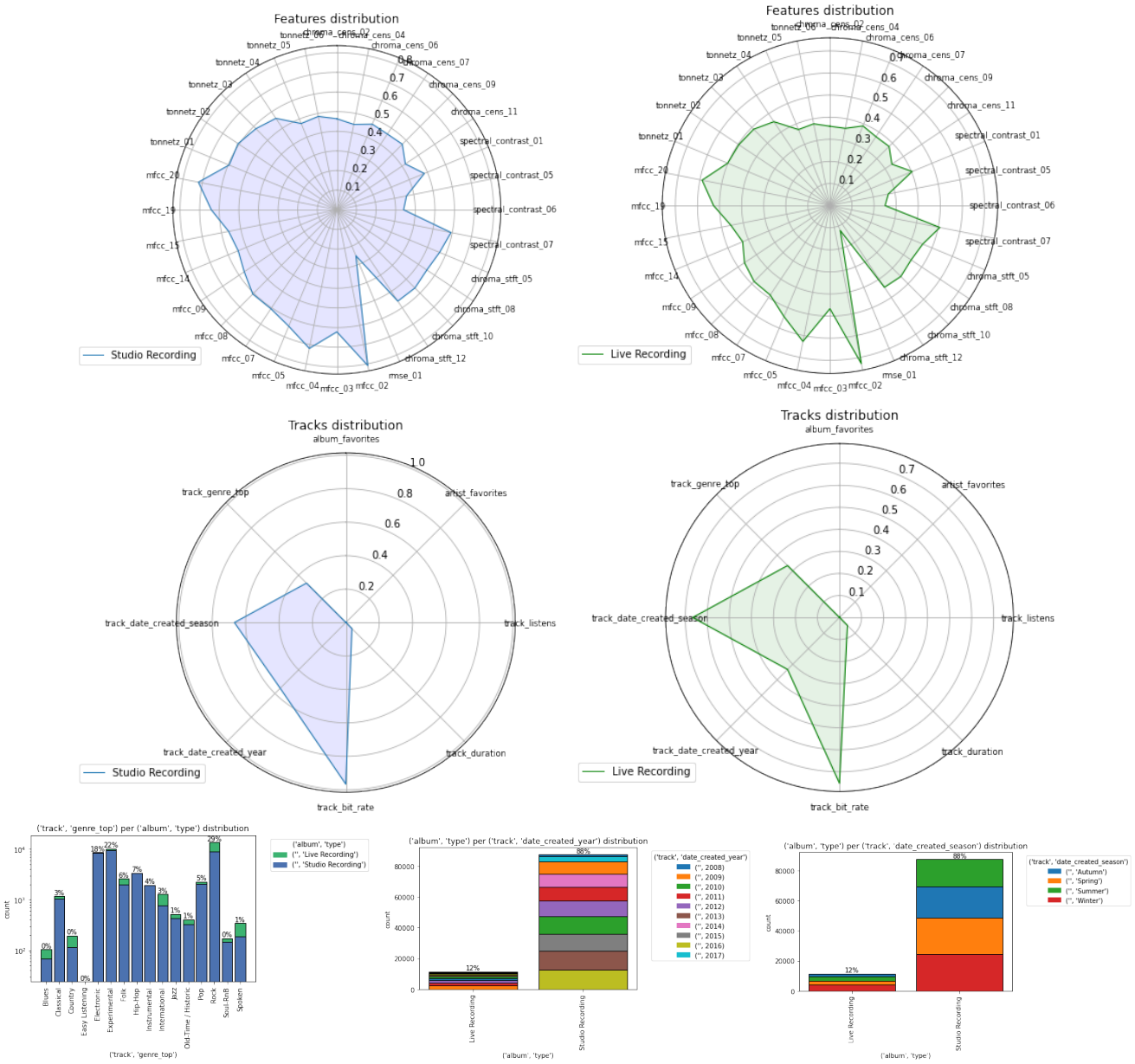


Figure 1: Target class labels' distributions.

1.2.3 Missing values and imputation

For missing albums and artists' metadata, we imputed them using given albums' or artists' other tracks' median or mode. If no albums' or artists' other tracks were present, we use column's median value as replacement. This is true for our target column *album_type*, too. We imputed missing album IDs, introducing the relative track in an other artist's album matching track title (replacing album's id and album's metadata with it, too)[1]. If no matching track title was detected, we crafted a new album ID, increasing the maximum album ID present in the dataset (22.940). We detected missing values in the form of *-1* and *0*[2], in *album_comments*, *album_tracks* and *track_number*, but since we didn't use these columns, we didn't proceed in imputing them. The same is true for 826 tracks having missing title, displaying 'Untitled', '(Untitled), (Title Unknown)', 'Title Unknown', '(title unknown)', 'title unknown', '(Untitled number)', 'Untitled #number' and other combinations as song's name. We observed conspicuous number of missing values in *artist_location*, *artist_latitude*, *artist_longitude* and *track_language_code* columns, too. Still, we tried to perform an imputing stage using a country map converter[3] and modified by hand to contemplate missing cases, but the pre-processing work was still not half-way finished, so we preferred not wasting additional time in the geographical columns' rescue. We also tried to guess *track_language_code* missing values, by noticing that *track_license*'s names, in many cases, ended with a country name. However, this

[1]This action is lawful since in the dataset are originally present artists having more version of the same song in the same album and since we got rid of *album_tracks*, *album_listens* and *track_number*, as we briefly will see.

[2]This last type of missing value is relative only to *track_number*, since track numbering starts from 1.

[3]The country map converter was retrieved from https://datahub.io/core/country-list.

imputation process lead to a zero-variance column in favor of the English language, so, at the end, we had to exclude *track_ language_ code*, too.

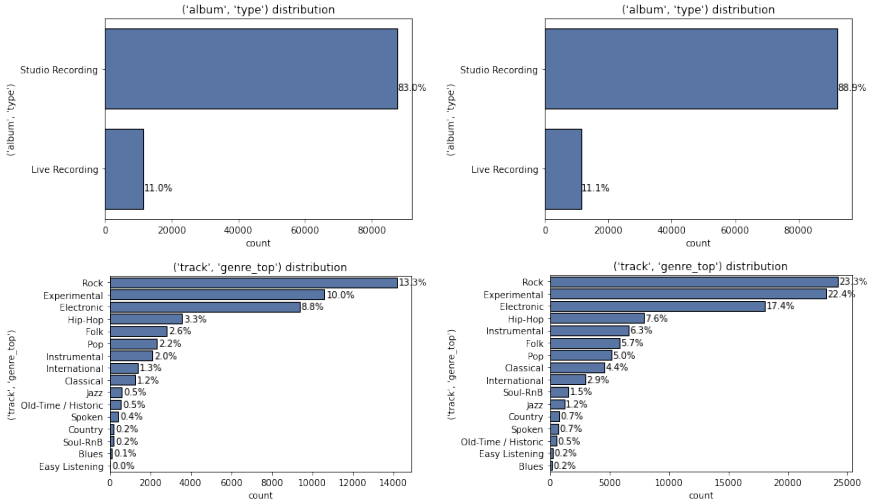


Figure 2: Before and after album_type and track_genre_top imputation.

To impute *track_ genre_ top* we use the following procedure: 1. we dated back a given track's missing genre, by looking and evaluating *track_ genres* column. In particular, we performed a majority vote among the genres present in the list; 2. if a tie showed in *track_ genres*, we performed the majority vote upon *track_ genres_ all*, which contains a super-set of the tracks present in *track_ genres*; 3. for tracks with ties among *track_ genres_ all*, too, or with empty genre lists, we performed a majority vote upon *album_ tags*, *artist_ tags* and *track_ tags*, taking into account matching tags referring to our 161 genre taxonomy (mapped afterwards in the usual 16 parent genres); 4. if all the above majority votes failed, by taking into account that an album usually has a limited amount of genres, we tried to match tracks with missing genres to tracks without the same missing data and present in the same album, using as matching factors, the bit-rate; 5. for tracks being part of a single track album (and so being solely in it) or not displaying matching bit-rate in the same album, we performed the same reasoning, but this time using artist's bit-rate; 6. if all the above steps failed, we merged all genre_top suggested by the previous five processes and performed a majority vote upon them; 7. if again, a tie showed, iterating over each genre involved in the tie, we used lowest difference among track's bit-rate and genre's bit-rate median value, to enact as a genre tie solver[4]. We used median values, to mitigate bit-rate's outliers, but if the tie still persisted, we used mean values; 8. whereupon, if again a ties showed, we used lowest difference among track's bit-rate and the 16 main genres' bit-rate mean value[5]. Lastly to impute *track_ bit_ rate* we used matching *track_ genre_ top* present in the same album or artist recordings, previously removing all tracks having *track_ genre_ top* set to NaN and *track_ bit_ rate* set to -1, since both columns are heavily imputed based on the match of these two columns. The major imputation performed can be observed in Figure 2.

1.2.4 Features filtering

We detected that 8.12% of albums have a number of tracks present in the dataset greater than what they stated, while 1.04% of albums have non-univocal track numbers. For this reasons we disregarded *album_ tracks* and *track_ number*, as being biased columns and moderate correlated among them (+0.53). We expected that album's listens gathered tracks' play-counts, adding to the counting tracks absent in the FMA collection, too. We actually found out than only 20.02% of albums shared this characteristic. This means that *album_ listens* and *track_ listens* assume a different semantic, in the sense they both are related to album's and track's popularity, but the two likeness aren't correlated. For this reason, we disregarded *album_ listens*, preferring to use more congruous features to detect albums' audience satisfaction. Regarding comments columns, since a comment can have a positive or negative meaning, we prefer to disregard them as audience' satisfaction toward albums/artists/tracks, since this sometimes can be misleading. To detect audience's appreciation, we preferred to take into account *album_ favorites*, *artist_ favorites* and *track_ listens*, since they always convey a positive reference (*track_ favorites* and *track_ interest* are positive correlate with *track_ listens* and so omitted, as shown in Figure 3 (left)). As regards date columns, *track_ date_ recorded* is a biased column and so we won't use it, since we detected many inconsistencies with the other columns (i.e. 151 albums being release before their tracks' recording, 6 artist beginning their career after their songs recording, 68 tracks being created on Echonest before their actual recording). We will omit

[4]Hypothesis: if a track's bit-rate is nearer to a genre_top A's median bit-rate than it is to a genre_top B's median bit-rate, it is more probable that the track belongs to A than B.

[5]We used directly the mean, since many genres shared the same median values.

album_ date_ released, *artist_ active_ year_ begin* and *artist_ active_ year_ end*, too, due too many missing values, keeping only *track_ date_ created* column, since all the other are positive correlated with it as shown in Figure 3 (right). Other features not mentioned and present in the *Tracks* dataset were disregarded due to too many missing values, zero-variability or due to non-informative variables (i.e. titles, names, tags, etc).

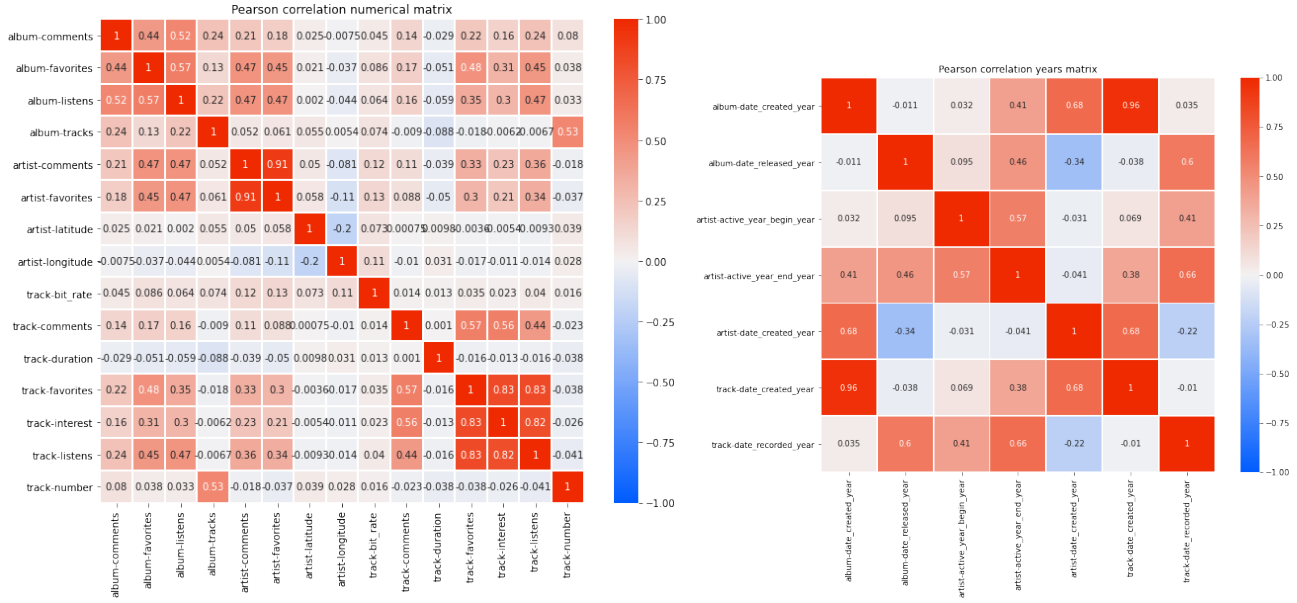


Figure 3: Pearson Correlation matrices.

As for the *Features* dataset, for each of the 74 predominant features we kept only the *mean* information. In the attempt to remove less features possible[6], we deleted 42 features, filtering them using the Pearson correlation. Please note that for space reason, the correlation matrix isn't provided. No correlation, however, was detected among *Features*'s variables and among our filtered *Tracks* and *Features* columns.
Our *Final dataset*'s features are divided as follows:

- **Categorical features**: *track_ genre_ top*, *album_ type*;
- **Ordinal features**: *track_ date_ created_ season*, *track_ date_ created_ year*[7];
- **Continous features**: *album_ favorites*, *artist_ favorites*, *track_ listens*, *track_ duration*, *track_ bit_ rate*, *chroma_ cens_ 02*, *chroma_ cens_ 04*, *chroma_ cens_ 06*, *chroma_ cens_ 07*, *chroma_ cens_ 09*, *chroma_ cens_ 11*, *chroma_ stft_ 05*, *chroma_ stft_ 10*, *chroma_ stft_ 12*, *mfcc_ 02*, *mfcc_ 03*, *mfcc_ 04*, *mfcc_ 05*, *mfcc_ 07*, *mfcc_ 08*, *mfcc_ 09*, *mfcc_ 14*, *mfcc_ 15*, *mfcc_ 19*, *mfcc_ 20*, *spectral_ contrast_ 01*, *spectral_ contrast_ 05*, *spectral_ contrast_ 06*, *spectral_ contrast_ 07*, *tonnetz_ 01*, *tonnetz_ 02*, *tonnetz_ 03*, *tonnetz_ 04*, *tonnetz_ 05*, *tonnetz_ 06*, *rmse_ 01*.

For DataFrame cross-referencing, we retained also *track_ id*, *album_ id*, *artist_ id* and *set_ split*, not used directly in our tasks. In the classification tasks we will perform more advanced feature selection methods, in the attempt to lower dataset's heterogeneous features, as explained in Section 1.4.

1.3 Outlier detection

Since we directly dealt with sophisticated outlier methods, in the pre-processing phase we didn't cleaned the dataset using the InterQuartile Range Method, due to the fact that we didn't detect the records suggested by the IQRM as being real anomalies[8]. For more advanced outlier detection methods, we used two *density-based* approaches (**LOF** and **DBSCAN**), a *model-based* approach (**ISOLATION FOREST**) a *distance-based* approach (**KNN**) and an *angle-based* approach(**ABOD**). Table 2 shows the number of outliers identified by each method. For DBSCAN, using the *knee method*, we detected the best initialization parameters as *eps=9* and *min_ samples=5*,

[6] *Features* dataset is made up by variables decomposed in sub-features, each one of them composing the global one. So we attempted to remove the smallest number of sub-features, to avoid removing global features' meaning. We achieved this intention, by exploiting the idea found at https://towardsdatascience.com/are-you-dropping-too-many-correlated-features.

[7] We split *track_ data_ created* column in the more informative *track_ data_ created_ season* and *track_ data_ created_ year*. The additional *track_ data_ created_ moment_ of_ the_ week* with values *Workday* (Monday-Friday) and *Weekend* (Saturday-Sunday) and *track_ data_ created_ moment_ of_ the_ day* with values *Morning* (06:01-12:00), *Afternoon* (12:01-18:00), *Evening* (18:01-00:00) and *Night* (00:01-06:00) were removed, since they resulted in zero-valued columns in favor of *Workday* and *Afternoon* values.

[8] These data points were rather tracks out of the norm, rather than outliers.

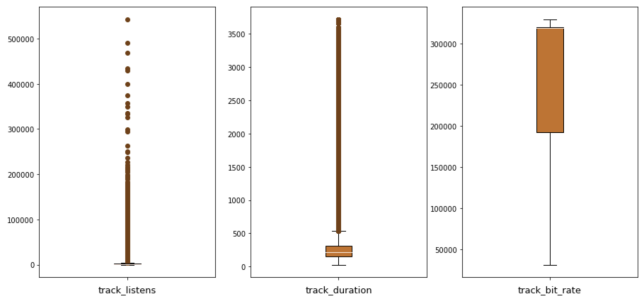


Figure 4: Boxplots.

Methods	N. outliers
LOF	3437
DBSCAN	281
KNN_5N	9296
KNN_10N	9794
ABOD_5N	10256
ABOD_10N	10079
ISOLATION_FOREST	142

Table 2: Outliers identified by different approaches.

which returned 281 noisy points, best compromise between under and overfitting. Regarding KNN's parameter tuning we followed the procedure undergone in Section 1.4.2, resulting in *n_neighbors = 10*. An equivalent procedure was used to tune ABOD'S *n_neighbors = 10* and LOF's *n_neighbors = 20* parameters. Looking at the scatterplot in Figure 6, we see that KNN identifies outliers better than ABOD and LOF. Table 3 shows, in the first part, the intersections between the first top 1% outliers for each method and, in the second part, the intersections between the top 10 outliers for each method, with the exception of the Isolation Forest, which didn't display any intersection with all the other approaches. It is important to note that the first 10 outliers of KNN_10N were also identified as such by the DBSCAN model.

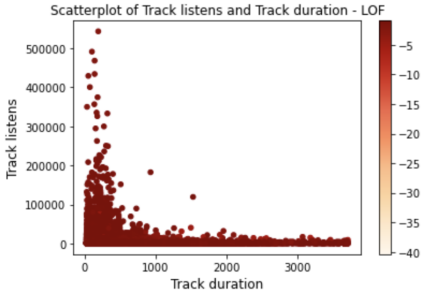


Figure 5: LOF 20N.

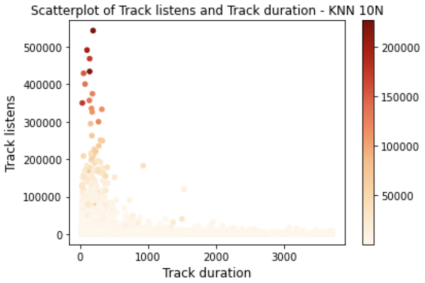


Figure 6: KNN 10N.

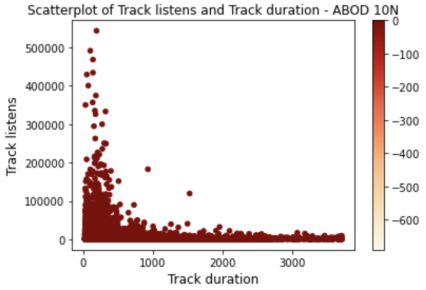


Figure 7: ABOD 10N.

Methods	∩ of top 1%	Methods	∩ of top 10
LOF - KNN_10N	148	:10_LOF - :10_KNN_10N	0
LOF - ABOD_10N	233	:10_LOF - :10_ABOD_10N	0
LOF - DBSCAN	2	:10_LOF - DBSCAN	0
KNN_10N - ABOD_10N	784	:10_KNN_10N - :10_ABOD_10N	8
KNN_10N - DBSCAN	78	:10_KNN_10N - DBSCAN	10
ABOD_10N - DBSCAN	66	:10_ABOD_10N - DBSCAN	9

Table 3: Intersections of top 1%, of top 10 and with DBSCAN.

Among all the anomaly methods, we created a pool of outliers to evaluate and to remove, according with the following considerations. *Live Recording* can last hours, so we didn't imposed an upper bound on *track_duration*. However, we removed the four longest tracks, since they were longer than 2 hours, while other tracks were less than one and a half hour. We performed a similar reasoning for *track_bit_rate*; we imposed a lower bound of 24.000 and upper bound of 41.6000, thus leading to 21 and 3 bit rate outlier tracks removal, respectively. Regarding *track_listens*, *artist_favorites* and *album_favorites*, extreme values are justified by audience's appreciation or not. However, we had to correct a track having set its listens to 0, by using other album track's listens mean. By proceeding with our outlier removal, we got rid of 1.918 records. Thus our *Final dataset*, taking into account also the previous sections and record removal, is made up by **103.798 records** and **44 columns** (**59 columns** when the categorical features are onehot encoded and the ordinal features are integer encoded.)

1.4 Basic Classifiers

The input data already provided a training and test splitting notion via the *set_split* column, so we recycled the data division, checking first if the target variable was well represented in all splits. The training and test composition, with which we worked, are summarized in Table 4.

X_train	X_test	y_train	y_test
(92.834, 55)	(10.874, 55)	(92.834, 1)	(10.874, 1)

Table 4: Train and test split.

Since the dataset still has a conspicuous number of features, to avoid the *curse of dimensionality*, beside the previous mentioned Pearson Correlation Method, we evaluated three different roads: (i) **Univariate Method**, using **SelectKBest** with the **ANOVA F-test**; (ii) **Feature Importance Method**, using **RFE** with features ranked according to *feature_importances_* or *coef_* derived from a **DecisionTreeClassifier** and a **LogisticRegression**, respectively; (iii) **VarianceThreshold**, removing features whose variance doesn't meet a **tuned threshold** equal to **0.01**[9]. Regarding models' hyper-parameter configurations on classification tasks, we evaluated them using a **3-fold cross-validation** via the **StratifiedKFold**. Since both class labels are equally important, we used the **f1_weighted** measure as **RandomizedGridSearch**'s scoring metric, aiming for a trade-off among precision and recall scores. All required stages were organized in a **Pipeline**, preceded by a **ColumnTransform** and whose use aided us to cope with additional tasks, too: (i) scaler selection, among **MinMaxScaler**, **StandardScaler**, **MaxAbsScaler** and **RobustScaler**; (ii) discretizer selection, evaluating different strategies (*uniform*, *quantile*, *kmeans*), number of bins (*range(2, 11)*) and encodings (*onehot*, *ordinal*) via the **KBinsDiscretizer**.

1.4.1 DecisionTreeClassifier

The best feature combination was directly provided by the classifier, while building the tree, resulting in **37 features** (top three: *track_bit_rate*, *mfcc_02* and *mfcc_4*). We tried to enhance model's performance using the above mentioned feature selection methods, without detecting performance improvements. Since Decision trees are able to cope with categorical variables, we first used all features. However, the best performance was obtained by using exclusively continuous variables, as shown in Figure 8. Model's hyper-parameter tuning can be observed in Table 5. From Figure 8, we can see how the model how the model is quite good in avoiding overfitting and as expected, shows a rectilinear decision boundary.

Table 5: Decision Tree hyper-parameters tuning.

Parameters	Description	Values	Best values
criterion	function to measure the quality of a split	gini, entropy	entropy
max_features	# of features to consider for best split	None, auto, sqrt, log2	log2
max_depth	max depth of the tree	range(1, n_features+1)	37
min_samples split	min samples required to split a node	range(10, 101, 10)	50
min_samples_leaf	min samples required to be a leaf node	range(10, 101, 10)	60
class_weight	weights associated with classes	None, balanced	None

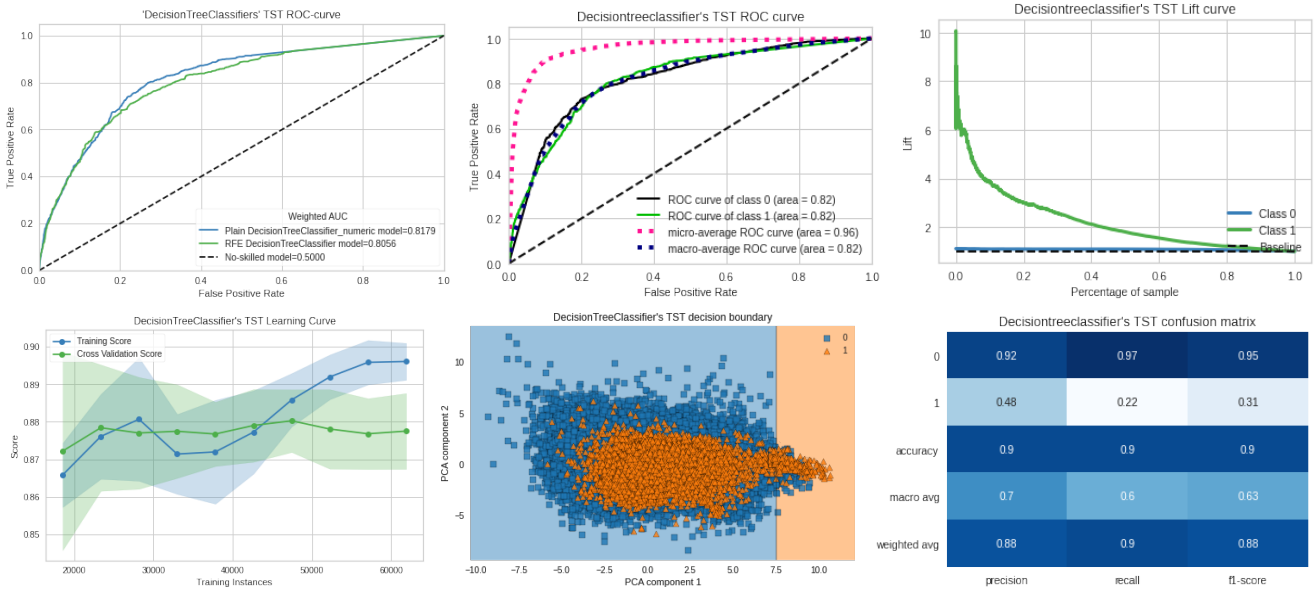


Figure 8: Best DecisionTreeClassifiers among continuous and mixed-type features performances (upper left) and absolute best DecisionTreeClassifier in-depth performance (upper middle and right, lower left, middle and right.).

[9]Setting a threshold equal to 0.01 lead us to retain 28 features, best compromise to removing all low frequency variables and keeping some discriminant ones.

1.4.2 KNeighborsClassifier

The best feature combination was provided by **ANOVA**, resulting in **31 features** (top three: *track_genre_top_Rock*, *track_date_created_year* and *mfcc_20*), while the more suitable scaler was **RobustScaler()**. We tried using all feature types, since we scaled the data and encoded categorical and ordinal features, thus resulting in slightly better performances than using exclusively continuous variables, as shown in Figure 9. Model's hyper-parameter tuning can be observed in Table 6. From Figure 9, we can see how the model is slightly overfitting data and shows an oval shaped decision boundary.

Table 6: KNN hyper-parameters tuning.

Parameters	Description	Values	Best Values
n_neighbors	# neighbors	range(3, $\sqrt{n\_records} * 2$, 2)	121
weights	prediction's weight function	uniform, distance	uniform
p	Minkowski's power parameter	1, 2	1

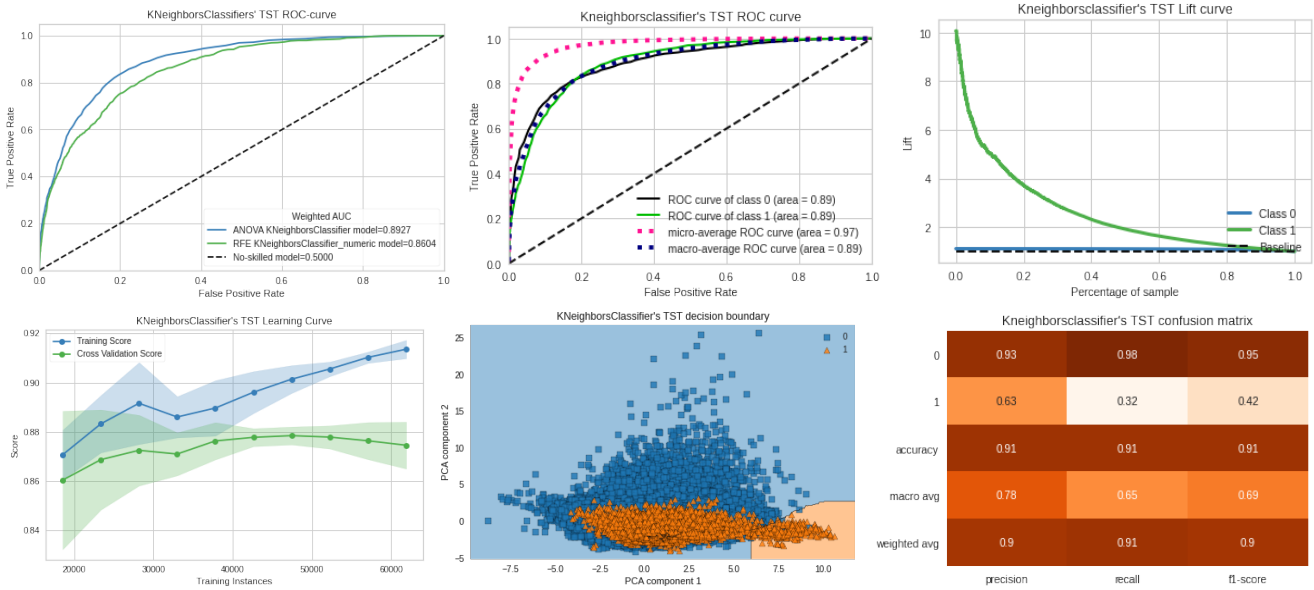


Figure 9: Best KNNs among continuous and mixed-type features performances (upper left) and absolute best KNN in-depth performance (upper middle and right, lower left, middle and right).

1.4.3 Final evaluations

Even if our KNeighborsClassifier slightly overfited the data, it was able to better discriminate both class labels. We believe that this is linked to its robustness in our data imbalance then our DecisionTreeClassifier. Thus, among our simple classifiers, **KNeighborsClassifier is the best**.Table 7 shows an anticipation of our trained, more advanced classifiers

	Accuracy	Precision	Recall	F1-score	AUC - ROC
DecisionTreeClassifier	0.90	0.88	0.90	0.88	0.82
KNeighborsClassifier	0.91	0.90	0.91	0.90	0.89
BernoulliNB	0.86	0.90	0.86	0.88	0.85
LogisticRegression	0.73	0.90	0.73	0.78	0.82
LinearSVC	0.72	0.90	0.72	0.78	0.83
MLPClassifier_1_layer	0.92	0.90	0.92	0.91	0.90
RandomForest	0.91	0.90	0.91	0.89	0.90
RIPPER	0.90	0.90	0.90	0.89	0.65

Table 7: Trained classifiers recap.

1.5 Imbalance

As we can see from Figure 10, our dataset presents a skew in the class labels distribution. We addressed this imbalanced problem using two approaches:

1. **Oversampling**: we oversampled the minority class, using **RandomOversampling**, **SMOTE**, **Borderline-SMOTE**, **SVSMOTE** and **ADASYN**;
2. **Undersampling**: we undersampled the majority class, using **RandomUndersampling**, **NearMiss**, **Tomek-Link**, **EditedNearestNeighbours** and **OneSidedSelection**.

Since our instance-based learner outperforms our eager learner, we tried to enhance DecisionTree's performance by applying the above, mentioned methods on it. As discovered from Section 1.4.1, DecisionTree works better using only continuous variables, so we performed the imbalance method upon them, having, however, the attention to scale the data, since many algorithms use intrinsic KNN, SVM and distance-base notions.

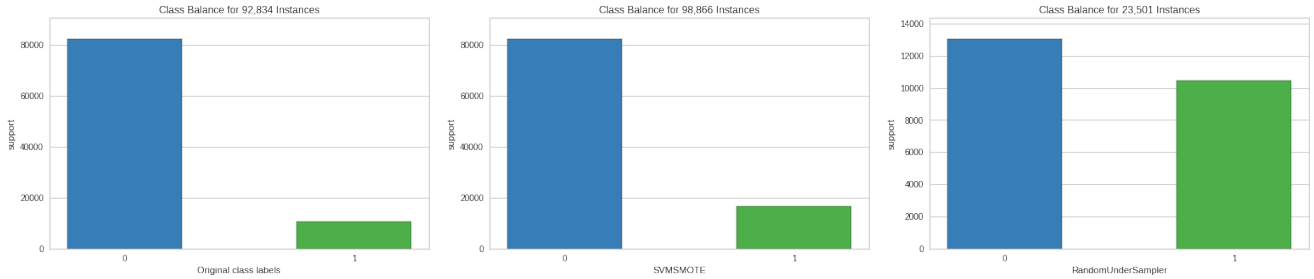


Figure 10: Original class labels balance (left), best Oversampling class labels balance (middle) and best Undersampling class labels balance (right).

1.5.1 Oversampling

This technique can be effective for models that seek good splits, such as our tested DecionTreeClassifier. Indeed, increasing minority class' frequency can make the model better fit the data. This, however, can lead to higher computational cost and overfitting. For this reason we were careful how and how many, new minority examples were synthesized. Models' hyper-parameter tuning can be observed in Table 8[10]. The best oversampler method is **SVSMOTE**, thus attempting to populate regions where there are fewer examples of the minority class.

Table 8: Oversamplers hyper-parameters tuning.

Parameters	Description	Values	Best Values
sampling_strategy	desired ratio samples' number	np.arange(0.1, 1.1, 0.1)	0.2
k_neighbors	neighbours to used to construct synthetic samples	range(1, $\sqrt{n\_records} * 2$)	582
m_neighbors	neighbours to use to determine if a minority sample is in danger	range(1, $\sqrt{n\_records} * 2$)	540

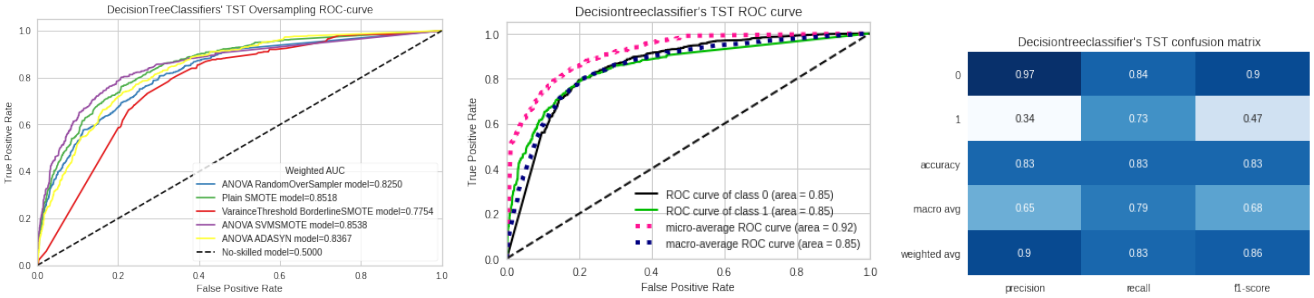


Figure 11: Oversamplers' performances (left) and SVMSMOTE performance (middle and right).

1.5.2 Undersampling

This approach is more suitable for those datasets where there is a class imbalance although still a sufficient number of discriminant, minority class examples, which isn't our case as we shortly we will see. Decreasing majority class' frequency, indeed, can worsen majority class detection, without bringing any benefit to minority class recognition. For this reason we were careful how and how many, majority class examples were deleted. Models' hyper-parameter

[10]Please note that not all algorithms required all parameters setting, but for space concern we tried to agglomerate them all in a single table.

tuning can be observed in Table 9. The best undersampler method is **RandomUnderSampler**, thus attempting to remove randomly examples of the majority class, reduces the number of important, majority class example removal and prevents TN score decline.

Table 9: Oversamplers hyper-parameters tuning.

Parameters	Description	Values	Best Values
sampling_strategy	desired ratio samples' number	np.arange(0.1, 1.1, 0.1)	0.8
n_neighbors	neighbours to used to construct synthetic samples	range(1, $\sqrt{n\_records} * 2$)	/
n_seeds_S	samples to extract to build the set S	range(200, n_label_0, 200)	/

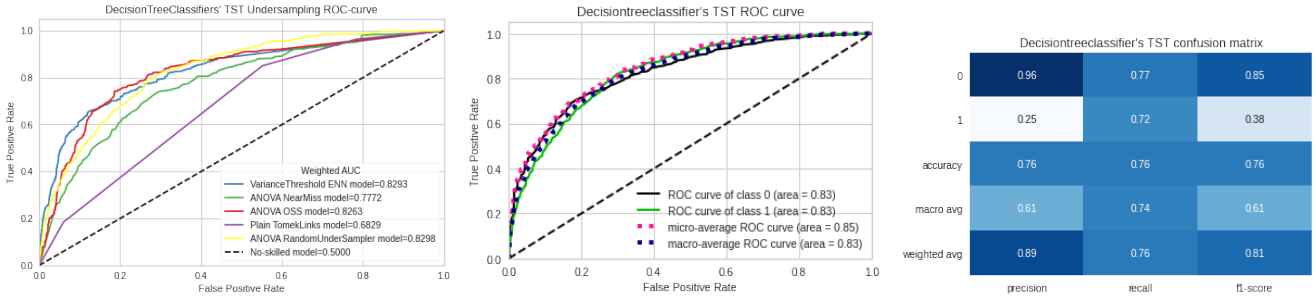


Figure 12: Undersamplers' performances (left) and RandomUnderSampler performance (middle and right).

1.5.3 Final evaluations

One of undersampling's limitation is that examples from the majority class are deleted that may be useful, important, or perhaps critical to fitting a robust decision boundary. In addition, not being provided with discriminant minority class examples, in our dataset, made the **oversampling imbalance methodology outstanding the undersampling imbalance one**.

2 Module 2

2.1 Advanced Classifiers

2.1.1 Naive Bayes

We trained three different Naive Bayes Classifiers: **GaussianNB** using only continuous variables, **CategoricalNB** using only categorical variables[11] and **BernoulliNB** using all features. The only hyper-parameter tuned was the Laplace smoothing parameter, as shown in Table 10. Both GaussianNB and BernoulliNB didn't benefit from our feature selection stages, while CategoricalNB appreciated VarianceThreshold's *track_ genre_ top* low frequency genre removal. The best performance was provided by BernoulliNB using **all features**, as shown in Figure 13. Since BernoulliNB expects Boolean's features, we had to onehotencode ordinal variables and to discretize and than to onehotencode continuous columns. The best binning procedure resulted in using an **uniform binning** with **10 bins**. From Figure 13, we can see how the model is good in generalizing and hasn't a clear decision boundary, due to the Maximum A Posteriori (MAP) estimation used by the algorithm.

Table 10: Naive Bayes hyper-parameters tuning

Parameters	Description	Values	Best Values
alpha	Additive (Laplace/Lidstone) smoothing parameter	range(1, 11)	1

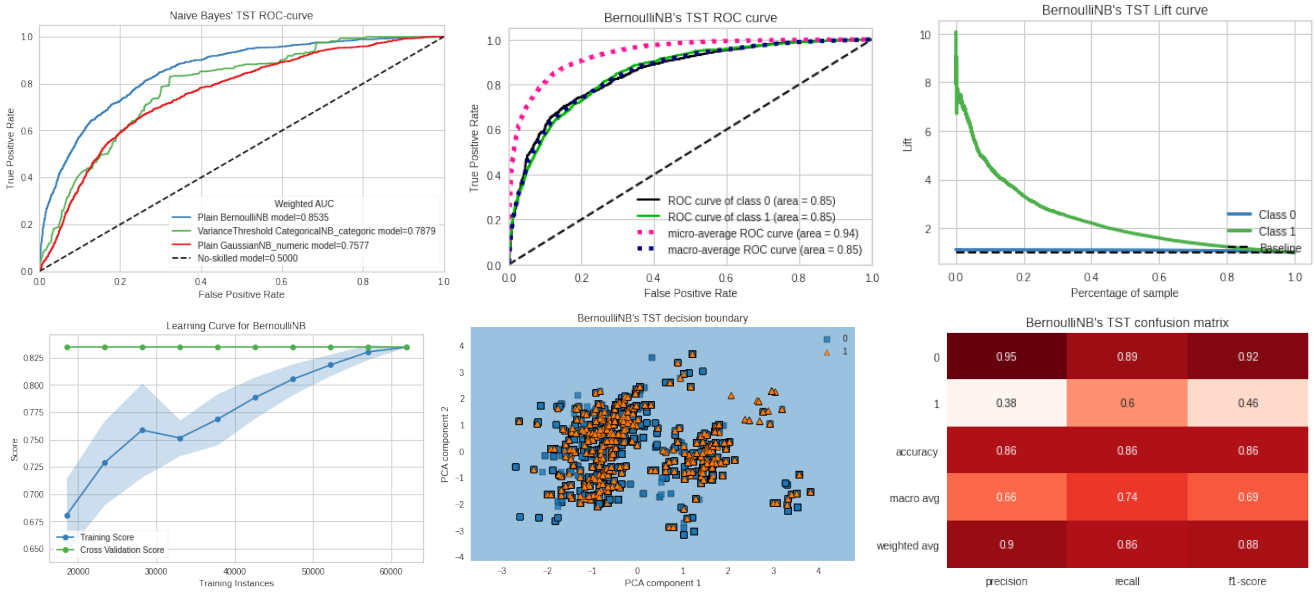


Figure 13: Best Naive Bayes performances (upper left) and absolute best Naive Bayes in-depth performance (upper middle and right, lower left, middle and right).

2.1.2 Logistic Regression

We trained three linear classifiers: **LogisticRegression**, **RidgeClassifier** and **SGDClassifier**. All models performed using continuous features, with comparable results between LogisticRegression and RidgeClassifier, as shown in Figure 14. Since, however, LogisticRegression had a slightly higher *roc auc*, we enact it as best model. Its best feature combination was provided by **ANOVA**, resulting in **37 features** (top three:*mfcc_ 20*, *mfcc_ 03* and *mfcc_ 07*), while the more suitable scaler was **RobustScaler()**. Models hyper-parameter tuning can be observed in Table 11. Since we deal with a large dataset, by default, we used *saga* solver. For SGD classifier we tuned also:

- *loss* (hinge, log, modified_huber, squared_hinge, perceptron; best: *modified_ huber*);
- *learning_ rate* (constant, optimal, invscaling, adaptive; best: *optimal*), alpha(1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0; best: *1e-05*)[12];
- eta0 (1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0, 0.0; best: *1.0*);

From Figure 14, we can see how the model is quite good in generalizing and as expected, shows a linear decision boundary. However, its balanced class weight choice, makes it less accurate, due to an increase in TP and FP score, as well.

[11] We onehotencoded ordinal features, too, to avoid using only *track_ genre_ top* features as single classifier trait.
[12] *alpha* was tuned for RidgeClassifier (best: *1e-03*)), too.

Table 11: Regression hyper-parameters tuning

Parameters	Description	Values	Best Values
C	inverse of regularization strength	0.001, 0.01, 1.0, 10.0, 50.0, 100.0	10.0
tol	tolerance for stopping criteria	1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-5
penalty	norm used in the penalization	l1, l2, elasticnet	l2
class_weight	weights associated with classes	None, balanced	balanced

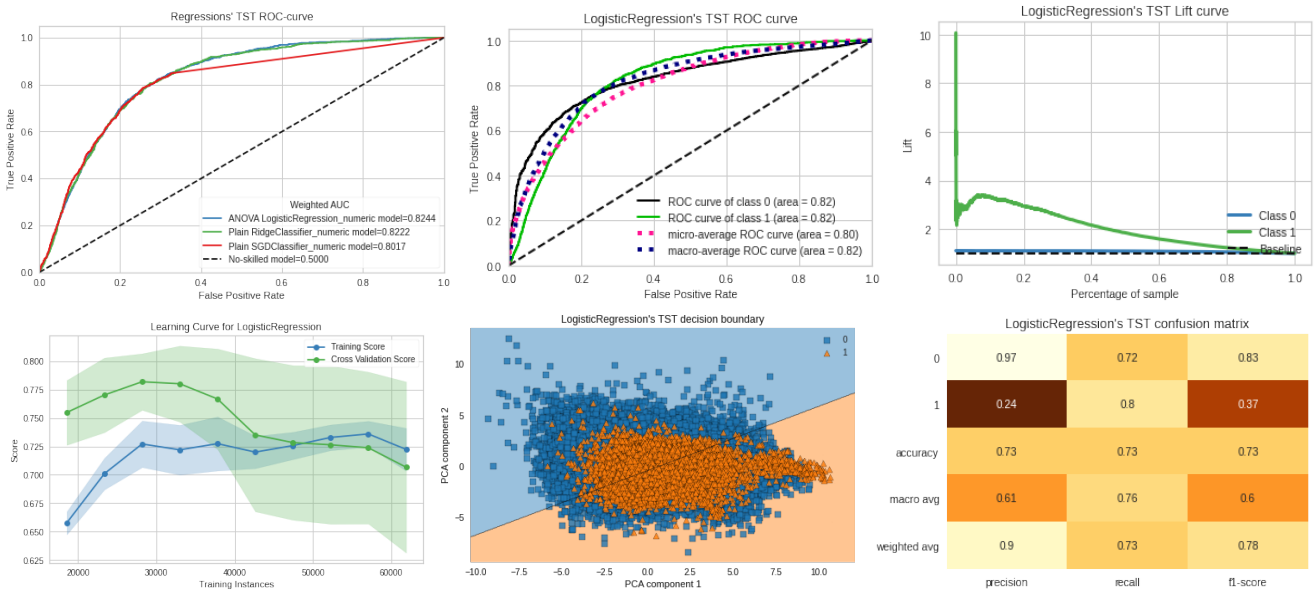


Figure 14: Best Regressors performances (upper left) and absolute best Regressor in-depth performance (upper middle and right, lower left, middle and right).

2.1.3 SVC

We trained four different types of kernels: **linear**, **poly**, **rbf** and **sigmoid**. All models benefit by using all features, filtered by the RFE, with the exception of the polynomial kernel, which preferred only continuous variables and no feature selection. Unlike our expectations, given our decision boundary's shape, the resulting best model didn't opted from the *Kernel trick*, but was again a linear class separator, thus **LinearSVC** being the best model in the family. The best feature combination was provided by **RFE**, resulting in **51 features** (top three:*track_genre_top_Old-Time_Historic*, *track_genre_top_Easy_Listening* and *track_bit_rate*), while the more suitable scaler was **MinMaxScaler()**. Models hyper-parameter tuning can be observed in Table 12. For non-linear kernels we tuned also:

- *gamma* (scale, auto; best: *auto*);
- *degree* (range(2, 13); best: *2*), solely the polynomial kernel.

From Figure 15, we can see how the model undergoes slightly overfitting when folds size increase and shows a linear decision boundary. Again, choosing to balance class weights, makes the model even less accurate, respect the LogisticRegression previously discussed.

Table 12: SVC hyper-parameters tuning

Parameters	Description	Values	Best Values
C	inverse of regularization strength	0.001, 0.01, 1.0, 10.0, 50.0, 100.0	10.0
tol	tolerance for stopping criteria	1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-5
class_weight	weights associated with classes	None, balanced	balanced

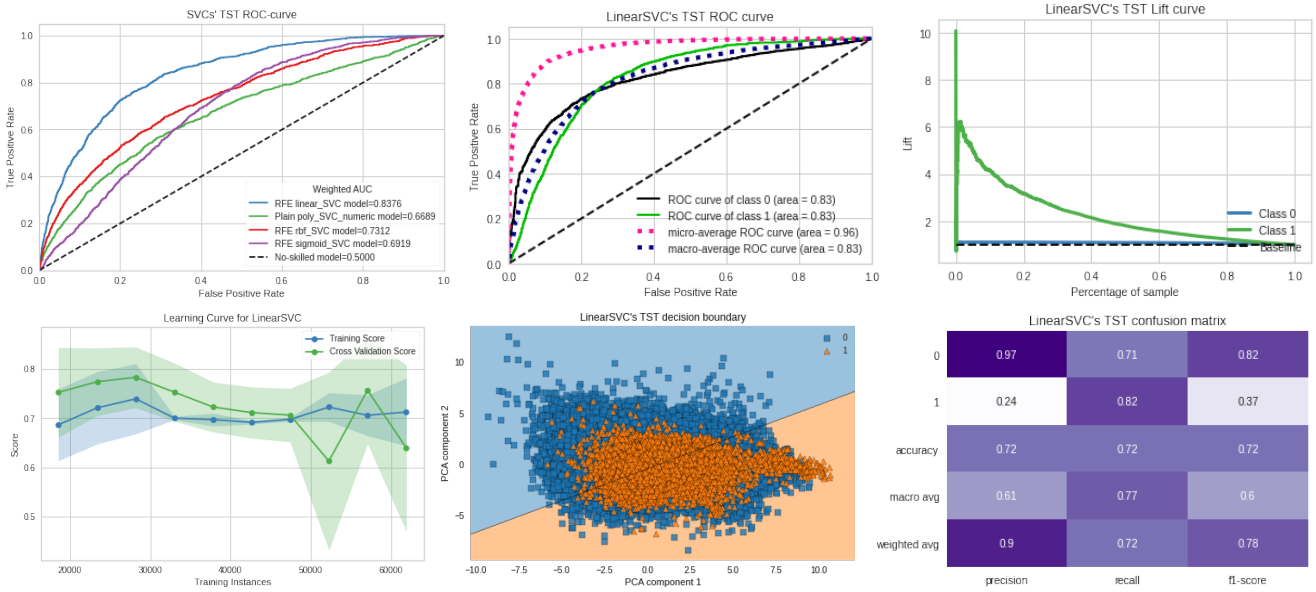


Figure 15: Best SVCs performances (upper left) and absolute best SVC in-depth performance (upper middle and right, lower left, middle and right).

2.1.4 Neural Networks

We trained three different types of network topologies: **Single Perceptron**, **MLPClassifier** and **KerasClassifier**, both using **1 hidden layer** and **2 hidden layers** for the last two[13]. As we saw from previous sections, our dataset seems mostly to benefit from a linear separation. For this reason we experimenting using one to two hidden layers, thus approximating any function that contains a continuous mapping from one finite space to another and an arbitrary decision boundary, respectively. Regarding units' number, we used between the size of the input and size of the output layers neurons. As we can observe from Figure 16, this prevented models having too few neurons (thus underfitting) or too many neurons (thus overfitting). We experimented with additional activation functions, optimizers and tuning parameter, as shown in Table 13, 14 and 15. On our Deep Neural Networks, we performed regularization using **L2** and **Dropout** on the visible and hidden layers[14]. Both 1 and 2 layer architectures benefit from a *1e-5* Dropout regularization and a *1e-3* and *1e-5* L2 regularization, respectively. This, however, didn't enhance KerasClassifier's performance. From Figure 16, we can see how the best architecture is the **1 layered MLPClassifier**, thus testifying again a preferable continuous, linear separability in our data. A part from the Single Perceptron, all other neural networks preferred using only continuous variables. In addition, all models benefit using **StandardScaler**. Networks' architecture are the following:

1. Single Perceptron: no-hidden layers;
2. MLPClassifier: (28,) and (12, 4);
3. KerasClassifier: (30,) and (26, 23);
4. KerasClassifier Dropout: (17,) and (27, 19);
5. KerasClassifier L2: (21,) and (4, 19).

[13]We made additional trials using more numerous hidden layers. However, we didn't detect any performance improvement, thus we stick with our simpler architectures.

[14]Both regularizations were tested in the range [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0].

Parameters	Description	Values	Best Values
penalty	norm used in the penalization	l1, l2, elasticnet	l2
alpha	regularisation multiplier	1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-5
tol	tolerance for stopping criteria	1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-1
class_weight	weights associated with classes	None, balanced	balanced

Table 13: Single Perceptron hyper-parameters tuning.

Parameters	Description	Values	Best Values
hidden_layer_sizes	# hidden layers	(i,), (i, j) for i, j in 1 to X_train.shape[1]	(28,)
activation	hidden layer activation	identity, logistic,tanh, relu	tanh
learning_rate	weight updates rate	constant, invscaling, adaptive	constant
solver	weight optimization solver	lbfgs, sgd, adam	adam
momentum	gradient descent update	list(np.arange(0.1, 1.1, 0.1))	0.1
alpha	regularisation multiplier	1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-1
tol	tolerance for stopping criteria	1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1.0	1e-4
class_weight	weights associated with classes	None, balanced	balanced

Table 14: MLPClassifier hyper-parameters tuning.

Parameters	Description	Values	Best Values
hidden_layer_sizes	# hidden layers	(i,), (i, j) for i, j in 1 to X_train.shape[1]	(30,)
optimizer	weight optimization solver	Adam, Adadelta, Adagrad, Adamax, Ftrl, SGD, RMSprop, Nadam	RMSprop
activation	hidden layer activation	relu, sigmoid, softmax, softplus, softsign, tanh, selu, elu, exponential	elu
nb_epoch	pass over entire	list(range(50, 501, 50))	350
batch_size	samples propagated	list(range(25, 251, 25))	50

Table 15: KerasClassifier hyper-parameters tuning.

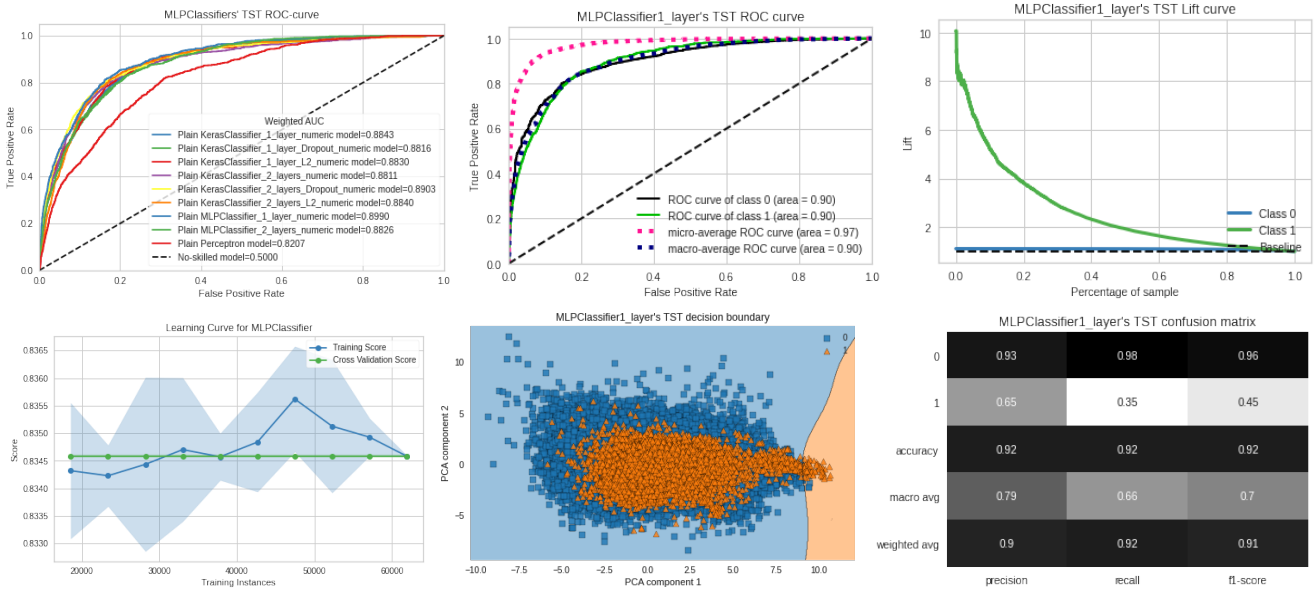


Figure 16: NNs' performances (left) and MLPClassifier performance (middle and right).

2.1.5 Ensemble Methods

We trained three types of meta estimators: **RandomForest** using the default *DecisionTreeClassifier*, **AdaBoost** using as base classifiers *DecisionTreeClassifier*, *GaussianNB*, *LogisticRegression* and **Bagging** using *DecisionTreeClassifier*, *KNeighborsClassifier*, *GaussianNB* and *LogisticRegression*. A part from Bagging's DecisionTree, all the other model preferred using only continuous variables[15] and mostly didn't used feature selectors. Among all ensemble methods, the most functional was RandomForest, which performed even better than the corresponding AdaBoost and Bagging. Actually, since RandomForest is an Boostrap Aggregation, we expected similarities with Bagging. Instead, it resemble more AdaBoost Decision tree's performances, with Bagging's tree being a standalone model in our trained family. Models hyper-parameter tuning can be observed in Tables 16, 17 and 18. From

[15] in Section 1.4.1, we saw how our tuned tree performed better using only numeric attributes, like our winner ensemble, RandomForest and the second placed AdaBoost decision tree, as we briefly we will see.

Figure 17, as expected, the decision boundary is rectilinear, with few positive splits, while feature scoring rely upon *chroma_ cens_ 02* and *chroma_ cens_ 04*. We can also see how the model, despite increasing CART's performance, prevents overfiting. This underlines how the *max_ samples* parameter tuning (% of samples to train each base estimator) is crucial to prevent this drawback.

Parameters	Description	Values	Best Values
n_estimators	number of trees in the forest	list(range(50, 501, 50)) + [25, 1000]	450
max_samples	% of samples to train each base estimator	list(np.arange(0.01, 0.11, 0.01))	0.06
class_weight	weights associated with classes	None, balanced, balanced_subsample	balanced

Table 16: RandomForest hyper-parameters tuning

Parameters	Description	Values	Best Values
n_estimators	number of base estimators	list(range(50, 501, 50)) + [25, 1000]	25
learning_rate	% of classifiers' contribution shrinking	list(np.arange(0.1, 1.1, 0.1))	0.9

Table 17: AdaBoost hyper-parameters tuning

Parameters	Description	Values	Best Values
n_estimators	number of base estimators	list(range(50, 501, 50)) + [25, 1000]	1000
max_samples	% of samples to train each base estimator	list(np.arange(0.01, 0.11, 0.1))	0.06

Table 18: Bagging hyper-parameters tuning

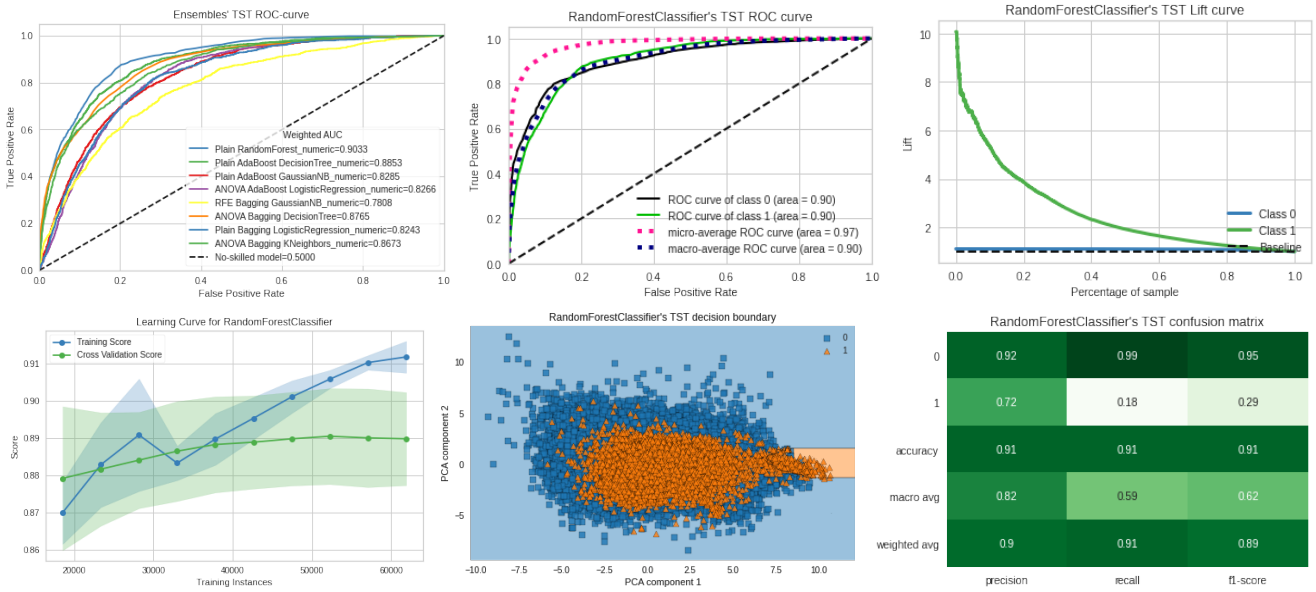


Figure 17: Ensembles' performances (left) and RandomForest performance (middle and right).

2.1.6 Ruled-based

We trained two types of rule-based estimators: **RIPPER** and **CN2Linear**. The first was trained using the RandomizedGridSearch. Second model's classification, instead, was implemented by us, matching each test record with the *IF...AND...THEN* rules produced by the training. We collected for each test record the prediction made by each matching rule, performing a *majority vote* to classify it. As shown in Figure 18, our trials didn't performed quite well. This has to be imputed to our imbalanced dataset and as a consequence to the generation of multiple negative class concerning rules. This phenomenon is accentuate for CN2, as shown by the *roc auc*, near resembling the dumb model's one[16]. Our best ruled-base model's (RIPPER) hyper-parameter tuning can be observed in Tables 19[17].

[16]This means that the model always predict the majority class. In fact, only 23 records were labeled as belonging to the minority class.

[17]We didn't provide a model's learning curve and decision boundary, due to the unsupported model type error when using *mlxtend* and *yellowbrick*, the two visual tool-kits we used to draw previous models' corresponding plots.

Table 19: RIPPER hyper-parameters tuning.

Parameters	Description	Values	Best Values
n_discretize_bins	discretisation bins	list(range(2, 11))	4
prune_size	pruning proportion	list(np.arange(0.1, 1.1, 0.1))	0.1
k	complexity stopping threshold	1, 2	1

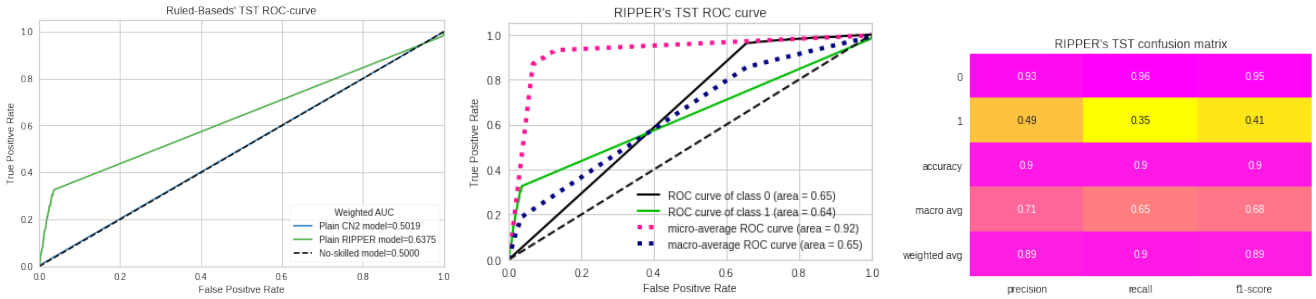


Figure 18: Rule-Baseds' performances (left) and RIPPER performance (middle and right).

2.1.7 Final evaluations

We just saw how our imbalanced dataset negatively impacted our Ruled-based classifiers. Regarding our basic classifiers, we detected how the KNeighborsClassifier is robuster than the DecisionTreeClassifer to the imbalance, resulting in one of the best performing models among all. Its high scores rely on parameters' fine tuning and the use of the Manhattan distance, less susceptible to noise. Outliers' influence are mitigated in our Neural Networks, by the refined tuned of neurons and layers, resulting in our absolute best performing model. Our Neural Networks, however, still doesn't perform well in scoring the same classification importance to both class labels, thus resulting in the highest score, by focusing mainly upon the negative examples. A similar behavior is hold by our RandomForestClassifier, with even lower positive record's prediction and second performing model placing. Higher increasing in the minority class classification can be seen in our BernoulliNB, LogisticRegression and SVCLinear, however at less precise and accurate predictions. Among the three, it is SVCLinear's lower performance that makes us think in a quite high *Maximal-Margin hyperplane* violation, thus leading our model in a *soft margin situation*, which lowers data variance, but increases bias. In addition, from model's learning curves, we learned that they are all quite good in generalizing, thus being robust in predicting unseen data. Finally, speaking about the decision boundary, we detected how among our two class labels it is blurred and overlapped, thus testifying how a track recorded in a studio or a recorded, live performance share blend characteristics.

2.2 Regression

For both types of linear regression (simple and multiple) we used **LinearRegression**, **Ridge** and **Lasso** methods. We considered the following continuous variables, since they received the highest scores in the previous classification tasks: *chroma_cens_02*, *chroma_cens_04*, *mfcc_02*, *mfcc_03*, *mfcc_04*, *mfcc_07*, *mfcc_20*, *spectral_contrast_05*, *spectral_contrast_07*, *rmse_01*, *tonnetz_02*, *track_bit_rate*, *track_duration*, *track_listens*, *album_favorites*, *artist_favorites*. For each regressor, every feature mentioned above was compared with all the others, both as an independent variable and as a dependent variable. The best result, in terms of R2, was obtained with the LinearRegression model, with *album_favorites* as independent variable and *artist_favorites* as dependent variable (**R2**: **0.487**, **MSE**: **3369.840**, **MAE**: **26.704**, **coef_** : **43.29**, **intercept_**: **30.98**).

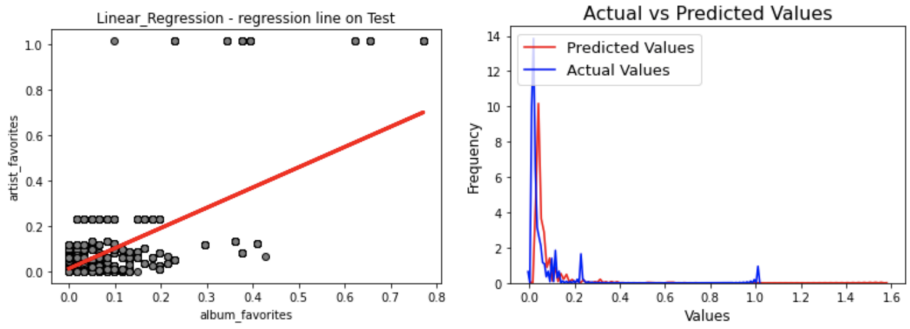


Figure 19: Simple regression (left) and multiple regression (right).

Since very similar results were obtained with the other two approaches, we decided to perform multiple linear regression using *artist_favorites* as the dependent variable. This time the best result was obtained with the Ridge

method, using only *album_favorites* and *track_listens* as independent variables (**R2**: **0.491**, **MSE**: **3338.206**, **MAE**: **26.229**, **coef_**: **[713.97 917.17]**, **intercept_**: **11.21**). Having added the variable **track_listens** to the independent variables led to a slight R2 improvement compared to that obtained by the simple linear regression.

3 Module 3

3.1 Time series data-set insight

Since our previous dataset contains the *track_date_created* column[18], we used it as guideline for our Time Series tasks. In particular, we articulated our analysis on two fronts: **one dataset following Studio Recording tracks creation** (thus following Studio Recordings' frequency through time) and **the other dataset following Live Recording tracks creation** (thus following Live Recordings' frequency through time). This methodology allowed us to discover intrinsic characteristics for each recording family, as well as making possible to compare similarities and differences among them, as we will briefly see. Track creation spans between 2008-11-25 17:49:06 and 2017-03-30 15:23:39, but marginal dates of such periods were deleted, to analyzed 8 complete years. So we analyzed the period spanning between **2009-01-01 00:00:00** and **2016-12-31 23:00:00**, reshaped by making all days having 24 hours and adding 186 missing days[19]. Since our *track_date_created* column has an *yyyy-mm-dd hh:mm:ss* time format, we summarized our two datasets by summing up **yearly**, **monthly**, **weekly**, and **daily** creations, thus retrieving different time granularity to further investigate. Finally, since our task were based upon distance notions, we evaluated two type of scalers, **TimeSeriesScalerMinMax** and **TimeSeriesScalerMeanVariance**, preferring the second for each task.

3.2 Motifs and anomalies

For our time series decomposition task, we had to performed firstly **Moving Average Smoothing** upon **weekly** ans **daily** granularities to remove the fine-grained variation between time steps and to better expose the signal of the underlying processes. Thus, we created three new series where the values were comprised of the average of raw observations in the original data, using a tuned **window width** equal to **5** and **21**, respectively. For motifs and discords discovery we slid different **sliding windows** across the time series, as summarized in Table 20.

	Y	**M**	**W**	**D**
sliding_windows	range(2, 5)	range(2, 13)	range(2, 53)	range(2, 91)

Table 20: Motifs' and discords' evaluated sliding windows.

Investigating each sliding window's output, we detected how yearly, retrieved motifs were too general to be real meaningful, while daily pattern was rather too specific to represent a Studio or Live Recording global characteristic. Monthly and weekly granularities, instead, were useful in making us discover an additive behavior for both recording types, as shown in Figure 21. Studio Recording tracks' creation are characterized by an increasing, cyclic trend, with two principal peaks detected in April-May and November-December and by an high activity plateau at each year's beginning. Live Recording tracks' creation are characterized by a decreasing, cyclic trend, with two main peaks in February-March and June-July. Our explanation is that Studio Recording peaks are linked to summer-hits' and Christmas ballads' release[20]. Live Recording's behavior, instead, can be justified by spring horse live performances (i.e. concerts), which take advantage of indoor facilities, while during the summer months, outdoor locations are the leading actors of its second peak of live enactments (i.e. festivals, fairs and in general an increased in the nightlife). Figure 22 and 23 show a zoom-in of these two conducts. Studio Recording's monthly granularity (Figure 22 (upper left)) specializes the trimester approaching the festive season (1^{st}, 8^{th}, 9^{th} and 10^{th} motif) and the summertime (2^{nd}, 3^{rd}, 4^{th}, 5^{th}, 6^{th} and 7^{th} motif). In particular, the quarters 2010-03-31 to 2010-05-31 and 2016-03-31 to 2016-05-31 present an activity's slow down in April, followed by a sudden increase in May. This is probably explained due to Easter's holidays (2010-04-04 and 2016-03-27, respectively). Studio Recording's weekly granularity (Figure 22 (upper right)), instead, captures the period between the two peaks, interspersed by a decreasing activity towards summer's end (1^{st} to 7^{th} motif).

[18] Since in the first two modules we retained only year's and season's creation date, we rustled up the original column.

[19] For our time series imputation process, we make use of the creation's inactivity notion, thus we filled time gaps with 0s to convey the idea that no creation took place, as we suspected happened during FMA author's data collection process.

[20] Usually these type of songs undergo a release's intensification just before summer's and wintry festivities' beginning.

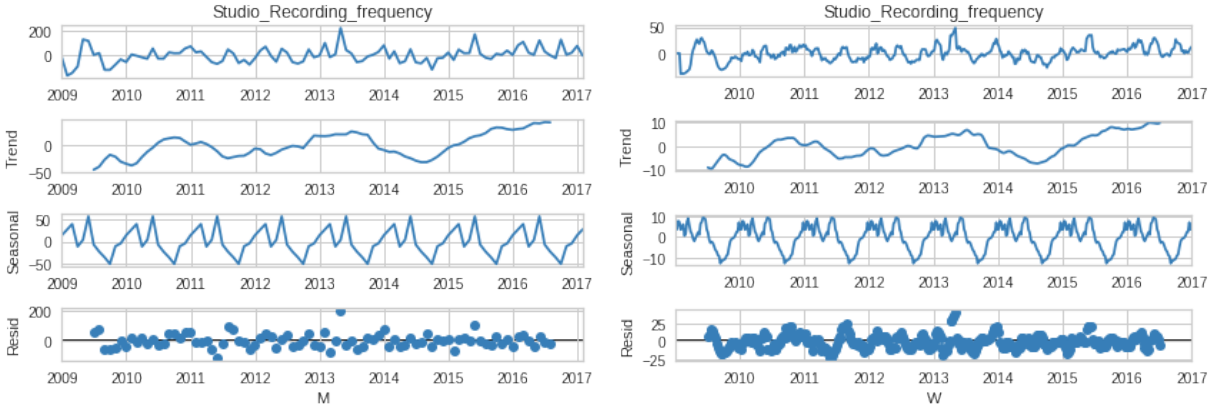


Figure 20: Studio Recording's monthly (left) and weekly (right) trend and seasonal.

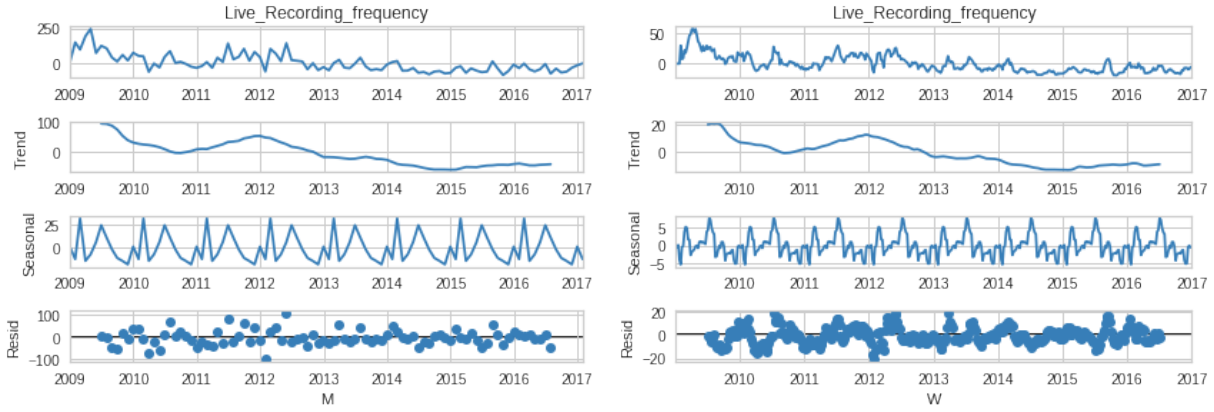


Figure 21: Live Recording's monthly (left) and weekly (right) trend and seasonal.

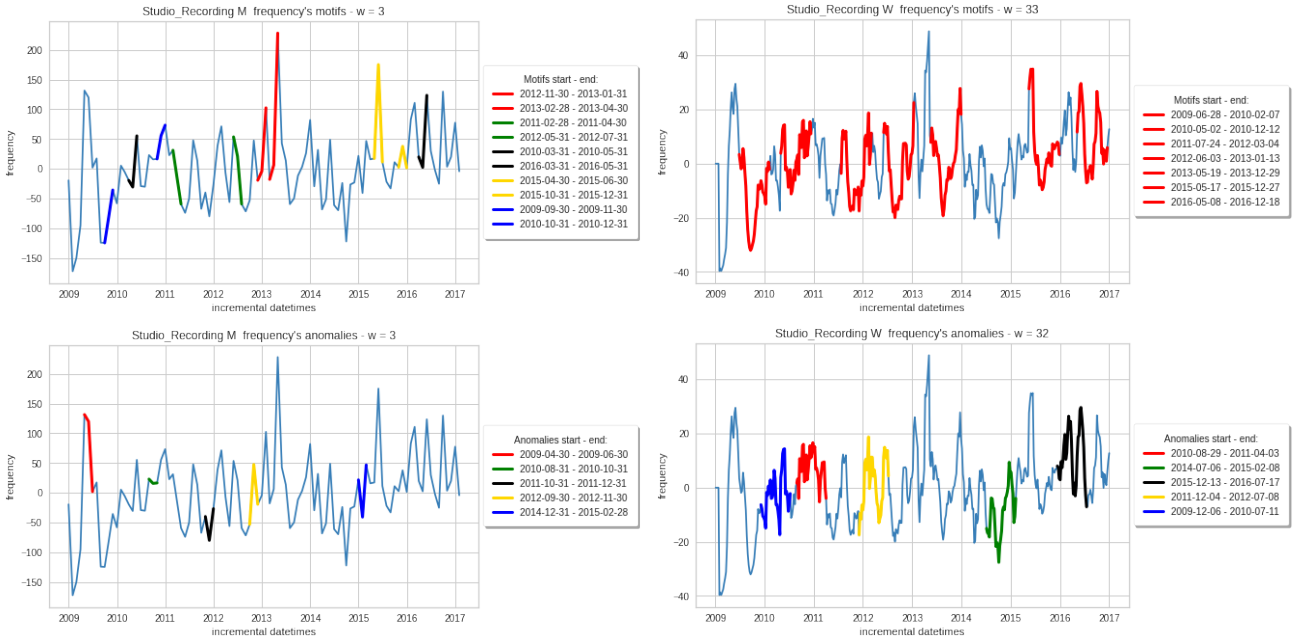


Figure 22: Studio Recording's motifs and anomalies.

Live Recording's monthly granularity (Figure 23 (upper left)) highlights, in the form of trimesters, summer peaks (1^{st} and 2^{nd} motif) and an activity slow down after February's and March's spikes (3^{rd}, 4^{th} 5^{th}, 8^{th} motif). Live Recording's monthly granularity (Figure 23 (upper right)) beholds a decreasing trend after spring's (1^{st}, 2^{nd}, 3^{rd}, 4^{th} motif) and summer's peaks (6^{th}, 7^{th}, 8^{th} motif). In additional, Live Recording spotted an activity decrease towards Christmas, due to less radio and live performances recorded in this period of the year.

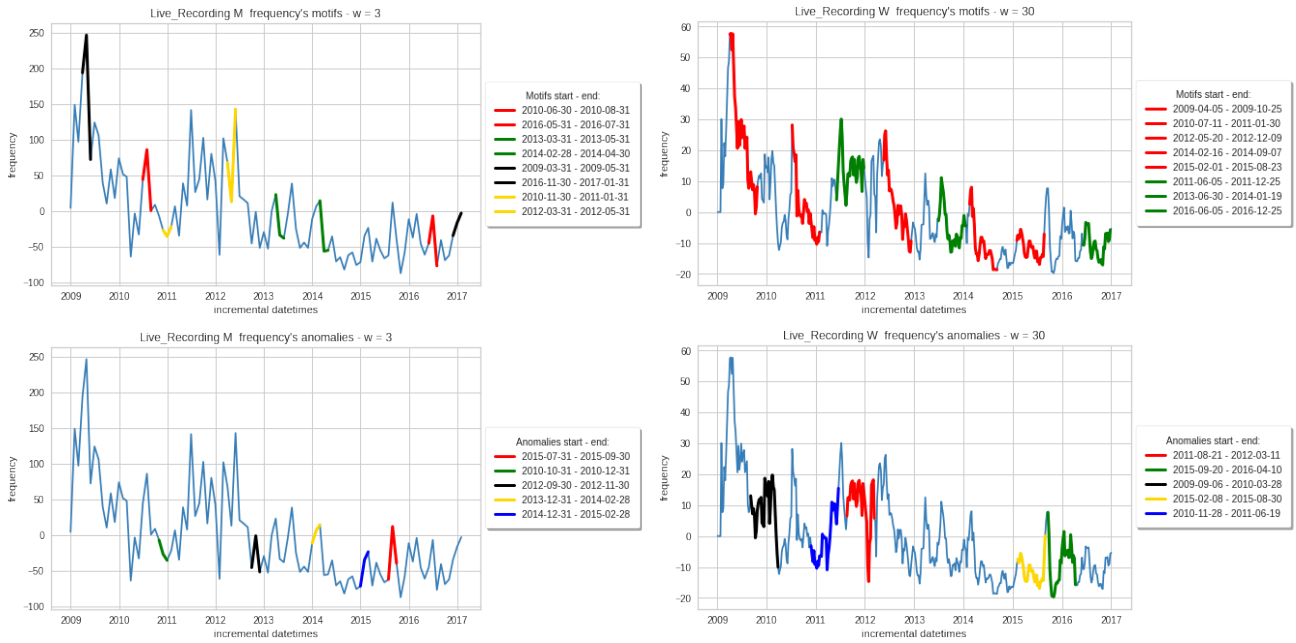


Figure 23: Live Recording's motifs and anomalies.

Studio and Live Recordings' discords are displayed in Figure 22 (bottom) and 23 (bottom), where rare trends were retrieved using an additional **exclusion zone** wide as sliding window's size, to prevent finding anomalies too close in time. Figure 22 (lower left) and Figure 22 (lower right) detect some exceptions to the rule with respect to what has been said so far about Studio Recordings: early releases during springtime, no activity reduction after summer or activity drops before or after the month of December. Figure 23 (lower left) and Figure 23 (lower right) project some delayed spring and summer peaks, as well as anomalous constant increasing or decreasing within the sliding window[21]. Finally, recalling Section 1.2.1, where we retained *track_ date_ created*'s years and seasons of creation, we confirm the following insights: 1. both Studio and Live Recordings were mainly created during the winter time. However, from the current section we learned that this is imputed to winter festivities approaching regarding Studio Recordings and late winter / early spring live performances regarding Live Recording; 2. always from the current section, we detected that other active months are April-May for Studio Recording and June-July for Live Recording; 3. generally speaking, Studio Recording undergoes an increasing performance over time, while Live Recording follows a decreasing trend through out the years, justified by more concise and numerous Studio Recording creations, opposite to more infrequent and less copious Live Recording creations.

3.3 Clustering

To create suitable datasets for our clustering and classification tasks, we additional divided each granularity on a year base. In this way we created 16 sub-time series[22], showing different time granularities to further test. We had only to dismiss the yearly granularity, since after the segmentation resulted in times series made up of a single timestamp. The resulting datasets' characteristics are shown in Table 21.

	M	**W**	**D**
shape	(16, 12)	(16, 52)	(16, 365)

Table 21: Time series classification and clustering datasets' shape.

Regarding the clustering task, we applied three different approaches: **shape-based**, **approximation-based** and **features-based**. For the first two, we used *TimeSeriesKMeans*, while for the last approach we used *KMeans*[23]. As approximations methods we used **Piecewise Aggregate Approximation**, **OneD Symbolic Aggregate Approximation** and **Symbolic Aggregate Approximation**, obtaining the best results by initializing window's size with half of the columns of its dataset. For the features-based approach we extracted the following pool of features for each time series: *med*, *10p*, *25p*, *50p* *75p*, *90p*, *iqr*, *cov*, *skw*, *kur*[24]. For all approaches, we evaluated SSE and silhouette performances, ranging k $\in [2, 15]$, whose best performances can be observed in Table 22.

[21] We just saw that both Studio and Live Recording time series have a camel case trend in time.

[22] 8 years following Studio Recording creations and another 8 years following Live Recording creations.

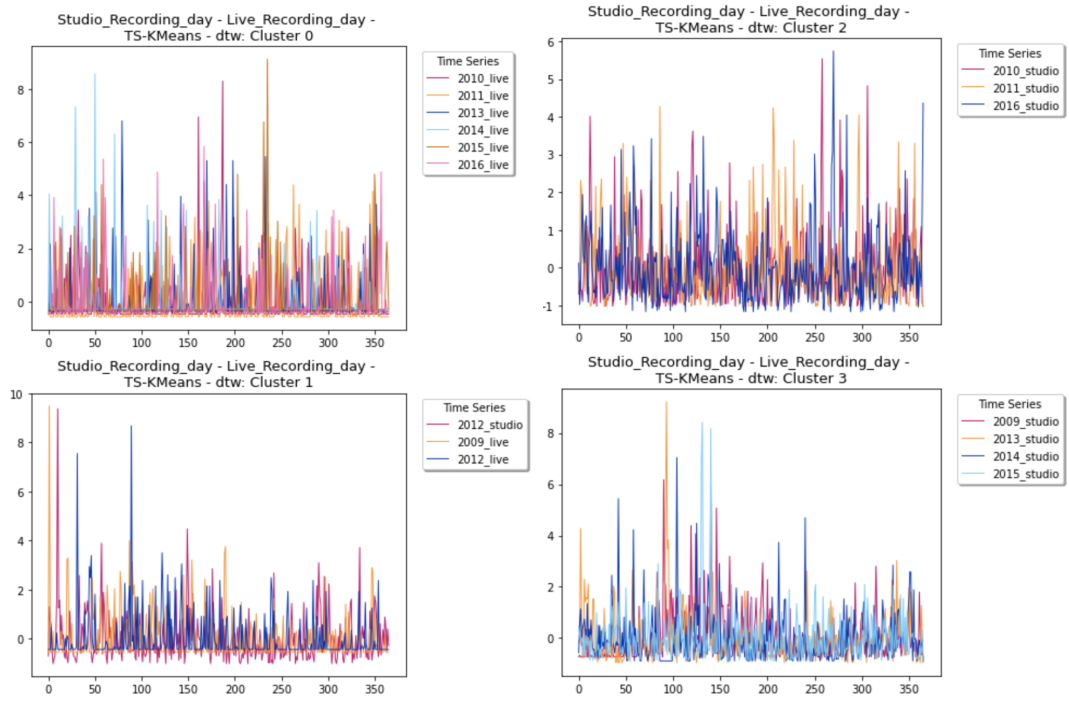
[23] For the clustering task we used *TSlearn*'s TimeSeriesKMeans and *Skealearn*'s Kmeans, thus being able to evaluate between the *euclidean* and the *dynamic time warping* metrics for the first and the *euclidean* distance for the second.

[24] Since we used the TimeSeriesScalerMeanVariance we omitted the zero-variance columns *avg*, *var* and *std*.

Clustering	Distance	Months			Weeks			Days		
		Best k	SSE	Silh	Best k	SSE	Silh	Best k	SSE	Silh
Shape-based	euclidean	6	3.94	0.18	8	19.07	0.07	7	182.01	0.04
Shape-based	dtw	3	2.65	0.19	5	10.25	0.07	4	66.83	0.07
Appr.-based PAA	euclidean	8	0.43	0.32	9	3.47	0.11	5	64.50	0.04
Appr.-based PAA	dtw	8	0.23	0.28	3	3.48	0.12	7	14.59	0.05
Appr.-based SAX1D	euclidean	5	10.88	0.15	5	109.26	0.06	7	8969.89	0.05
Appr.-based SAX1D	dtw	7	6.36	0.12	6	57.46	0.04	3	4382.36	0.06
Appr.-based SAX	euclidean	6	2.79	0.33	9	36.91	0.09	3	14516.83	0.05
Appr.-based SAX	dtw	4	2	0.30	3	33.23	0.12	8	1901.62	0.07
Features - based	euclidean	3	6.39	0.38	3	16.68	0.71	4	33.95	0.69

Table 22: Best k, SSE and silhouette score of different approaches (scaler: *TimeSeriesScalerMeanVariance*).

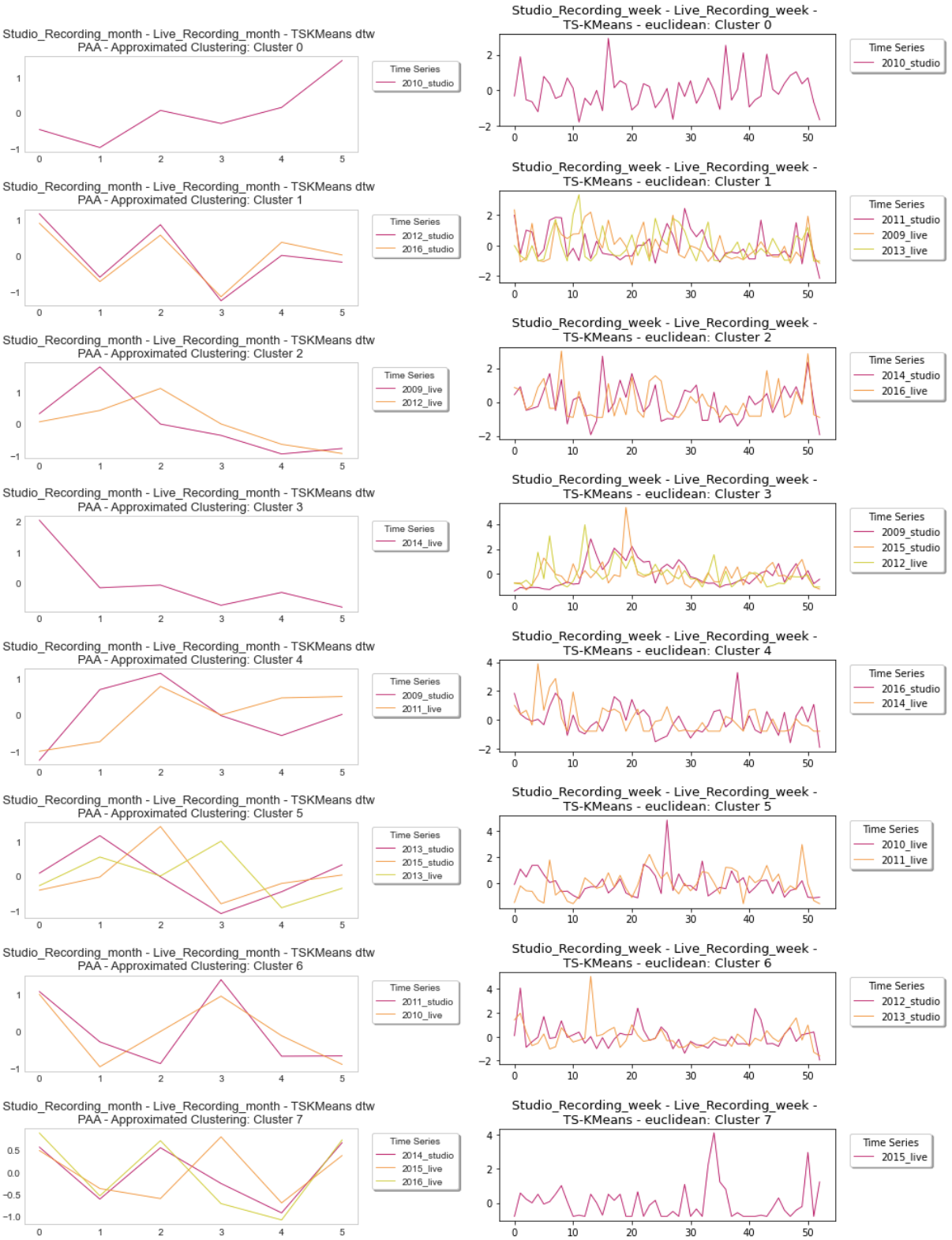
From an in-depth analysis of the obtained clusters, we observed that our most specific temporal granularity (daily granularity) benefited our clustering process, as the algorithm manages to find satisfactory result in terms of clusters' entropy and purity, with respect to the other two granularities (months and weeks). Thus a greater amount of information is decisive in separating Studio and Live recordings. To corroborate what has been said, Figure 24 shows the best obtained clustering: **shape-based TimeSeriesKMeans** with **dynamic time warping**. It identifies 4 clusters among which 3 out of 4 show an entropy equals to zero and a purity equals to one.



	Studio_Recording	Live_Recording	Tot_Cluster	Entropy	Purity
Cluster 0	0	6	6	-0.0000	1.0000
Cluster 1	1	2	3	0.9183	0.6667
Cluster 2	3	0	3	-0.0000	1.0000
Cluster 3	4	0	4	-0.0000	1.0000
Total	8	8	16	0.1722	0.9375

Figure 24: Best clustering (granularity: days).

Clusters' composition highlights how in cluster 0 are present almost all *Live Recording*'s time series, having peaks in the first part of the year and in the middle; in cluster 1, together with two *Live Recordings*, we find a *Studio Recording* time series, which, despite belonging to a different category, shows a similar trend to the other two components, i.e. peaks in the first part of the year. In clusters 2 and 3 we find only time series belonging to *Studio Recording* but, while in the cluster 2 we observe peaks in many periods of the year (in particular in the last part), in cluster 3 we observe peaks only close to the half of the year. As regards months and weeks, Figure 25 shows the obtained results. Their scores are good but not excellent, due to data's excessive generality.



	Studio_Recording	Live_Recording	Tot_Cluster	Entropy	Purity
Cluster 0	1	0	1	-0.0000	1.0000
Cluster 1	2	0	2	-0.0000	1.0000
Cluster 2	0	2	2	-0.0000	1.0000
Cluster 3	0	1	1	-0.0000	1.0000
Cluster 4	1	1	2	1.0000	0.5000
Cluster 5	2	1	3	0.9183	0.6667
Cluster 6	1	1	2	1.0000	0.5000
Cluster 7	1	2	3	0.9183	0.6667
Total	8	8	16	0.5944	0.7500

	Studio_Recording	Live_Recording	Tot_Cluster	Entropy	Purity
Cluster 0	1	0	1	-0.0000	1.0000
Cluster 1	1	2	3	0.9183	0.6667
Cluster 2	1	1	2	1.0000	0.5000
Cluster 3	2	1	3	0.9183	0.6667
Cluster 4	1	1	2	1.0000	0.5000
Cluster 5	0	2	2	-0.0000	1.0000
Cluster 6	2	0	2	-0.0000	1.0000
Cluster 7	0	1	1	-0.0000	1.0000
Total	8	8	16	0.5944	0.7500

Figure 25: Best clustering (granularities: months(left) and weeks(right)).

3.4 Classification

Recalling Table 21 datasets, our classification task translated in *correctly classifying time series' recording type*, based upon a yearly recorded data, further specialized in our usual **monthly**, **weekly** and **daily** granularities. We divided each dataset in a **90-10 stratified Train-Test Split**[25], thus we trained models using 14 time series and test their generalization performance upon the remaining 2 time series. For **both classes the year to test was 2015**, as choose by Sklearn's stratified *train_test_split*. Table 26 summarized the used classifiers. Hyper-parameters tuning followed DecisionTreeClassifier's (Table 5) and KNeighborsClassifier's (Table 6) procedure, with the additional *dtw* metrics for our KNeighborsTimeSeriesClassifier. For our ShapletClassifier, we firstly tuned its *shaplet size*, *shaplet number* and *regularizer's weight*.

	Parameters
ShapletClassifier	n_shapelets_per_size: {73: 2}, weight_regularizer:0.0001
Shaplet-distances-based DT	min_samples_split: 2, min_samples_leaf: 4, max_features: auto, max_depth: 2, criterion: gini, class_weight: balanced
Feature-based DT	min_samples_split: 6, min_samples_leaf: 2, max_features: log2, max_depth: 6, criterion: gini, class_weight: balanced
KNeighborsTimeSeriesClassifier	n_neighbors: 3, metric: dtw, weights: uniform

Figure 26: Time serie classifiers' best daily parameters.

Both monthly and weekly granularities didn't performed well in terms of classification, since all models were biased towards discovering shaplets/features resembling Live Recording time series. As we saw in our clustering task, indeed, a monthly or a weekly granularity generates impure clusters, meaning that exists years sharing a stronger similarity with inter-class' years than with its other intra-class' years. This implies that is quite difficult for these two to types of granularities to generate an accurate decision boundary among Studio and Live time series. Increasing the time granularity on a daily base, instead, allows to scale back time series' inter-class similarities, thus boosting up slightly time series' intra-class trends. Using a **daily granularity**, in fact, made us obtained a **100% accuracy prediction** using all our trained classifiers. Please note, however, that to reach this performance almost all data is used in the training phase, meaning that a clear demarcation between the two classes is still difficult. This remark is testified also by Tables 27, 28, 29 and 30, where Shaplet 1's assignment to Studio Recording class and Shaplet 2's assignment to Live Recording class it is given by 4 out of 7 train time series being nearer to them. Thus, both type of recordings have years withholding both shaplets, making a real distinction challenging.

	2009	**2014**	**2010**	**2016**	**2015**	**2013**	**2011**	**2012**
start	2009-07-29	2014-10-09	2010-05-07	2016-05-11	2015-06-17	2013-08-19	2011-06-08	2012-03-06
end	2009-10-10	2014-12-21	2010-07-19	2016-07-23	2015-08-29	2013-10-31	2011-08-20	2012-05-18
dist	7.7101	7.8099	8.01620	8.3589	8.4076	8.5817	8.9913	9.0860

Figure 27: Studio Recording time series' increasing distances from Shaplet 1.

	2014	**2016**	**2013**	**2015**	**2009**	**2012**	**2011**	**2010**
start	2014-02-10	2016-10-17	2013-07-14	2015-09-09	2009-09-27	2012-05-21	2011-08-29	2010-09-09
end	2014-04-24	2016-12-29	2013-09-25	2015-11-21	2009-12-09	2012-08-02	2011-11-10	2010-11-21
dist	7.8046	7.8547	8.1115	8.1438	8.2735	8.5335	8.8778	9.1528

Figure 28: Live Recording time series' increasing distances from Shaplet 1.

	2010	**2014**	**2009**	**2015**	**2013**	**2011**	**2016**	**2012**
start	2010-08-30	2014-09-14	2009-04-25	2015-01-03	2013-07-19	2011-06-04	2016-05-01	2012-09-13
end	2010-11-11	2014-11-26	2009-07-07	2015-03-17	2013-09-30	2011-08-16	2016-07-13	2012-11-25
dist	7.0947	7.3812	7.6311	8.5349	8.6660	8.7888	8.9110	9.0785

Figure 29: Studio Recording time series' increasing distances from Shaplet 2.

[25] According to our 70-30, 80-20, 85-15 and 90-10 rials, only the last hold-out allows to have a sufficient number of time series to train the ShapletModel and to find characteristic shaplets representing both classes. Otherwise, given the modest annual inter-class variance, most of the found shaplets belonged to class 1, i.e. the most stable class over time.

	2015	2009	2016	2014	2013	2012	2010	2011
start	2015-05-02	2009-09-30	2016-09-15	2014-08-21	2013-06-12	2012-05-24	2010-06-22	2011-09-15
end	2015-07-14	2009-12-12	2016-11-27	2014-11-02	2013-08-24	2012-08-05	2010-09-03	2011-11-27
dist	7.4816	7.5184	7.5388	7.7218	8.5217	8.6614	8.6756	8.9566

Figure 30: Live Recording time series' increasing distances from Shaplet 2.

The two retrieved shaplets are present in Figure 31, while thier best aligment on train and test are shown in Figure 33. They both underlines two different activity patterns of creation, with Shaplet 1 showing a rhythmic trend in time and Shaplet 2 a more irregular one. Usually Shaplet 1's cadenced activity involves workdays before, during and after the summertime, underlying how intense changes in the Studio, musical panorama and, probably, Echonest employee's holiday need a meticulous uploading planning. What has just been said is true for Live Recording too, this time with a more concise scheduling before Christmas. Shaplet 2 doesn't show a clear pattern, but conveys the idea of an isolated time series's portion, characterized by an irregular activity with a more pronounced peak, usually located before summer (Studio Recording), during summer (Live Recording) and approaching festivities (both).Thus, even if expressed with a different granularity, Section 3.2's motifs are confirmed by our shaplets, too. The years less represented by the shaplets are 2010, 2012 and 2012.

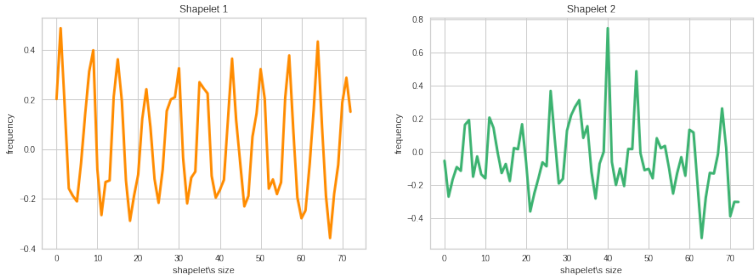


Figure 31: Shaplet 1's and Shaplet 2's trend.

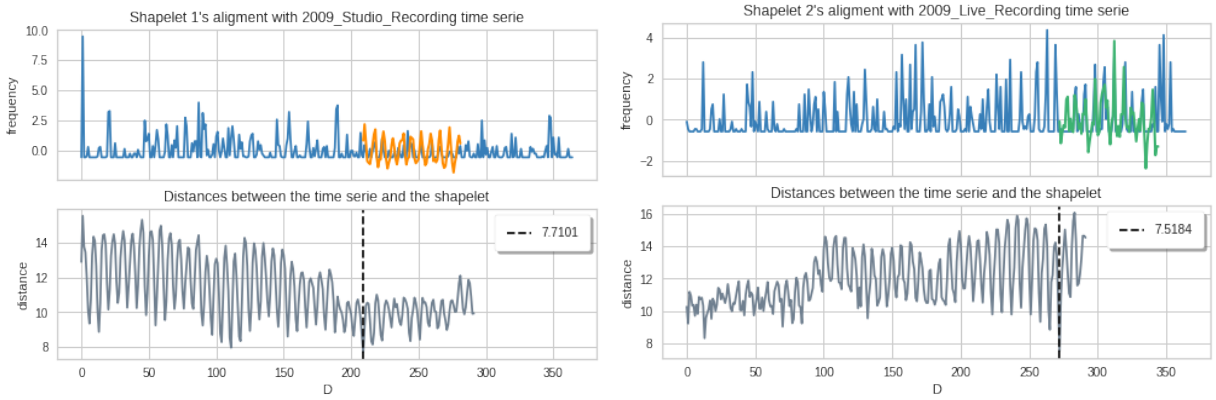


Figure 32: Shaplet 1's and Shaplet 2's best alignment on train.

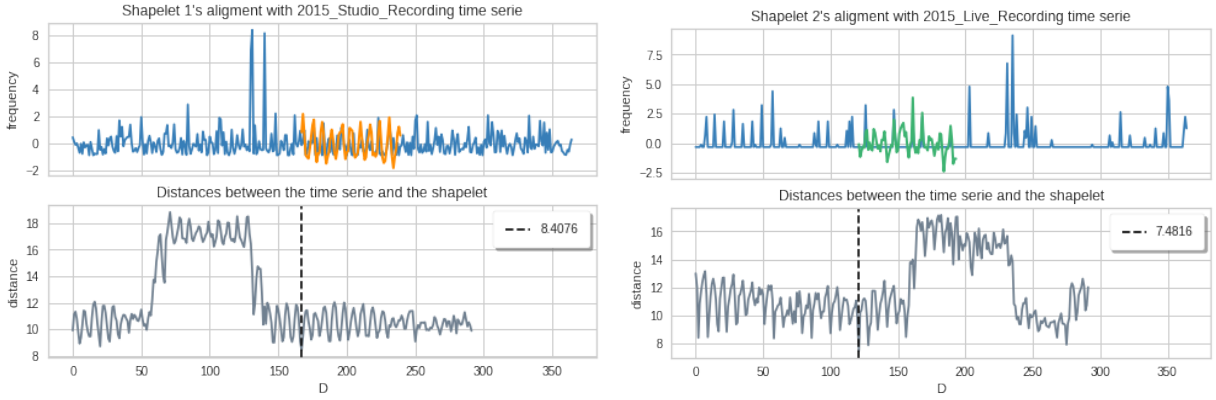


Figure 33: Shaplet 1's and Shaplet 2's alignments on test.

Finally, both distance-based and feature-base[26] DecisionTreeClassifier managed to classify the time series using two stumps, as shown in Figure 34, while KNeighborsTimeSeriesClassifier benefited from dtw's alignment between sequences.

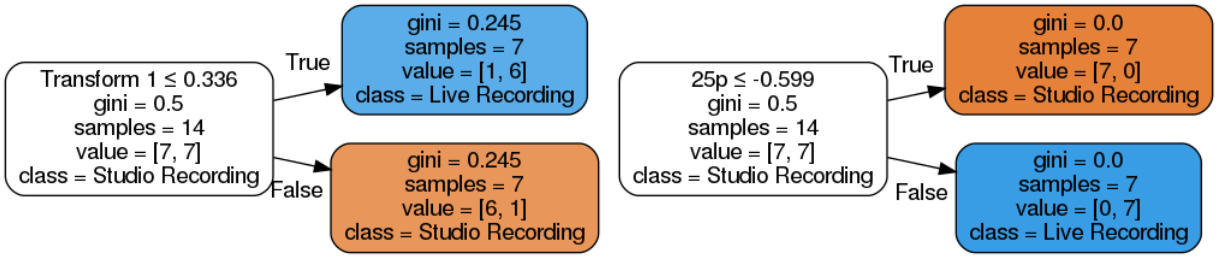


Figure 34: Distance-based and feature-base DecisionTreeClassifiers.

4 Module 4

4.1 Sequential Pattern Mining

Since in Section 3.2, we detected how monthly and weekly granularities were meaningful in detecting motifs, we reused them to mine sequential patterns in time. However, for space constrain and since no relevant patterns were retrieved using the weekly granularity[27], in the present report we are showing only the former one. In addition, we stuck with our Studio_Recording and Live_Recording time series distinction, to better characterize our two album types. We divided the two time series into 8 equal parts (one for each year), obtaining **two data sequence databases**, each one made out of **8 transactions**, having each sequence composed by **12 singleton events** (one for each month of the year)[28]. Finally, to convert the time series into a discrete format, we used the tuned approximation methods displayed in Table 23. For **both recording types**, the most suitable approximation in detecting relevant patterns was **SAX**.

album_type	PAA	SAX	ONE-D-SAX
Studio_Recroding	n_segments = 32	n_segments = 32, alphabet_size_avg = 4	n_segments = 32, alphabet_size_avg = 4, alphabet_size_slope = 4
Live_Recording	n_segments = 48	n_segments = 48 alphabet_size_avg = 6	n_segments = 48, alphabet_size_avg = 6, alphabet_size_slope = 6

Table 23: Time series' approximation parameters over the M time granularity.

As expected with min_supp threshold's increasing, sequential patterns' length (number of elements) decreases for both album types. In particular, regarding Studio Recordings an interesting length motif arises, since only 3, 6 and 12 adjacent (min_gap=1) elements are discovered among all supports, as shown in Figure 35. By using a min_sup>=4[29], in Figure 36, instead, we can notice the following sequential patterns:

- <**(2,), (2,), (2,)**> (support=6): it captures the renewal of Studio tracks at the end (2012, 2013, 2015, 2016) and beginning (2011, 2012, 2013, 2015) of most years;
- <**(1,), (1,), (1,)**> (support=5): it captures a moderate activity in 2010's and 2014's beginning and 2011's, 2013's and 2015's July-August-September trimester;
- <**(3,), (3,), (3,)**> (support=5): it captures Studio's March-April-May peaks (2009, 2013, 2015, 2016) and an activity increasing approaching 2010's Christmas;
- <**(0,), (0,), (0,)**> (support=4): it captures a low activity in 2009's beginning, 2009's, 2012's and 2014's July-August-September trimester and 2011's March-April-May quarter.

[26] We used the same features mentioned in the clustering section.
[27] As already seen in Section 3.2, **weekly** granularity specialize the more concise **monthly** granularity's motifs.
[28] Unfortunately both databases weren't meant to properly apply sequential pattern mining, due to the fact that in Section 3.1, we solely analyzed album creation's frequency through time, being compelled in the current task to use single, frequency, itemized events.
[29] Half of transactions present in the data sequence database.

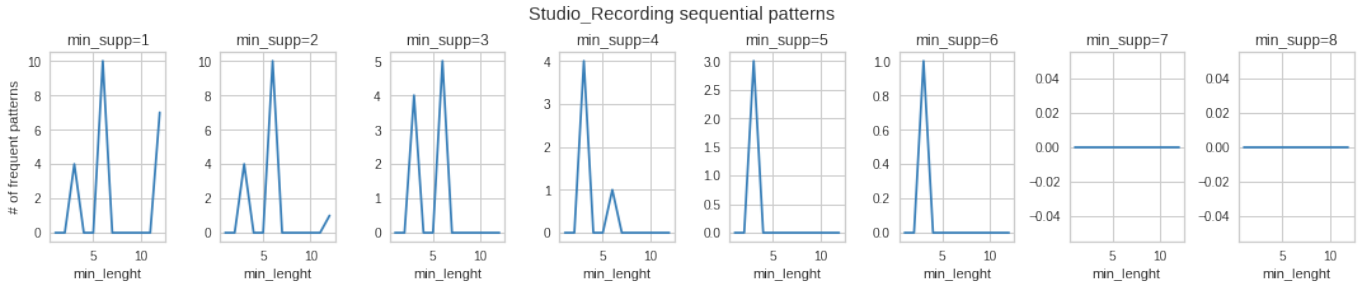


Figure 35: Studio_Recording SAX sequential patterns' min_supp.

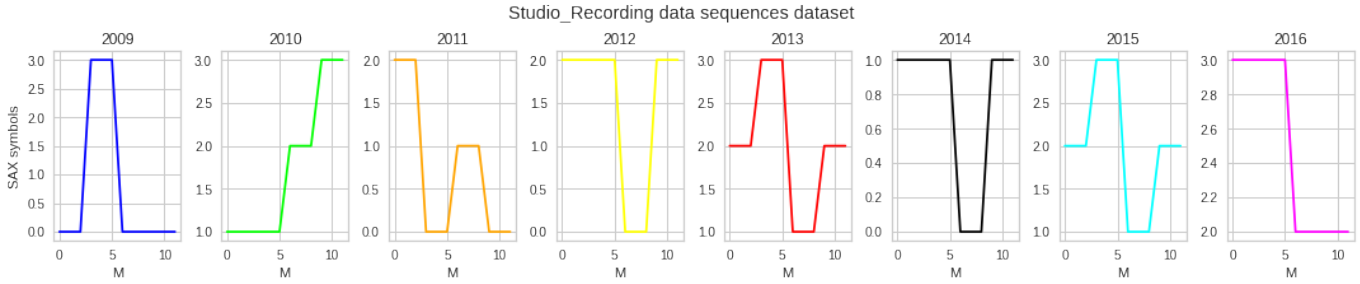


Figure 36: Studio_Recording SAX sequential patterns.

Live sequential patterns are made out of 2, 4, 6, 8 or 12 elements, as shown in Figure 37. From Figure 38, instead, we can notice how 2009 is approximately a stationary year[30] By using a min_sup >= 4, we detected the following sequential patterns:

- <**(1,), (1,), (1,), (1,)**> (support=5): it captures 2014's and 2015's spring, moderate activity, due to Easter's approaching, 2013's slowing down due to Christmas time and 2010's and 2016's mixture of the two set backs;
- <**(2,), (2,), (1,), (1,)**> (support=4): it captures 2016's degrading activity at the beginning of the year, followed by a moderate recovery in May-June and a go along summer decrease and 2012's, 2013's and 2015's summer-autumn diminish approaching Christmas;
- <**(3,), (3,), (1,), (1,)**> (support=4): it captures 2014's degrading activity at the beginning of the year and 2010's, 2012's and 2013's slowing down approaching Christmas.

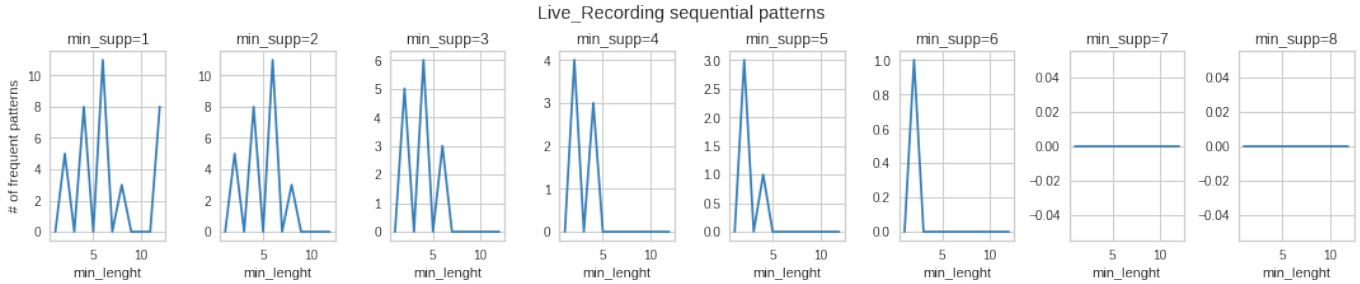


Figure 37: Live_Recording SAX sequential patterns' min_supp.

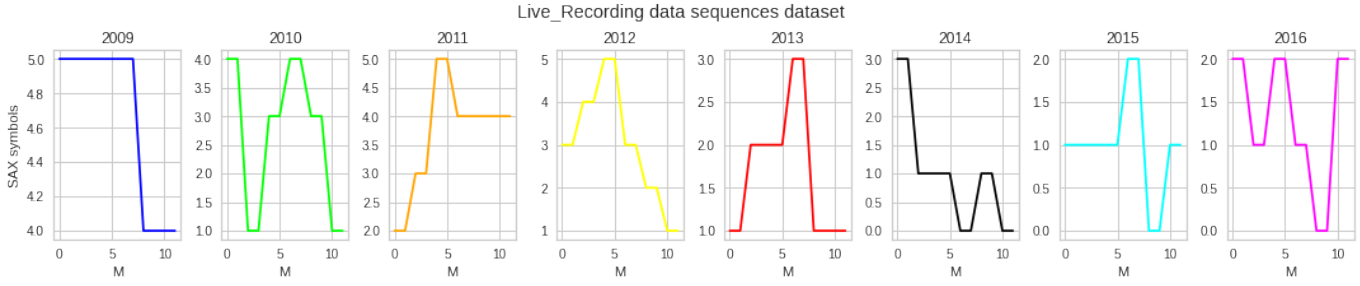


Figure 38: Live_Recording SAX sequential patterns.

[30]This pattern is in compliance with what already observed in Section 3.2, thus that the most active, Live_Recording year is 2009, followed afterwards, by a decreasing trend.

As a conclusion, both for Studio and Live time series', the majority of the motifs retrieved in Section 3.2 are confirmed by SPM, too.

4.2 Advanced Clustering

As advanced clustering methods we ran **X-Means** and **OPTICS**, evaluating each combination of features described in Section 2.2[31]. Although using this feature selection approach to mitigate the *curse of dimensionality* and to enhance records' discrimination, both our clustering methods suffered the overlapping decision boundary among our two class labels, detected in Section 2.1. This indicates that **the dataset is not suitable for clustering**, even if using distance-based and density-based advanced clustering techniques. Finally, we scaled our continuous variables using **RobustScaler**, in the attempt to mitigate, as possible, noisy points.

4.2.1 X-Means

X-Means' best feature combination resulted in: ***mfcc_03***, ***track_listens***, ***artist_favorites***.
We set *amount_initial_centers* = *2* (amount of clusters from which X-Means will start its analysis), following our ideal division of the space of points in two labels: Studio_Recording and Live_Recording. Regarding the *kmax* (maximum number of clusters that can be allocated), we set an upper bound in the [2, 50] range, resulting in **kmax = 3** as the best value, both for metric scores and interpretability, as shown in Table 24 and Figure 39. Finally, as metric we used the **euclidean distance**, since X-Means relies upon K-Means[32].

N_clusters	SSE	Silhouette
3	12010.2659	0.9188

Table 24: X-Means best result.

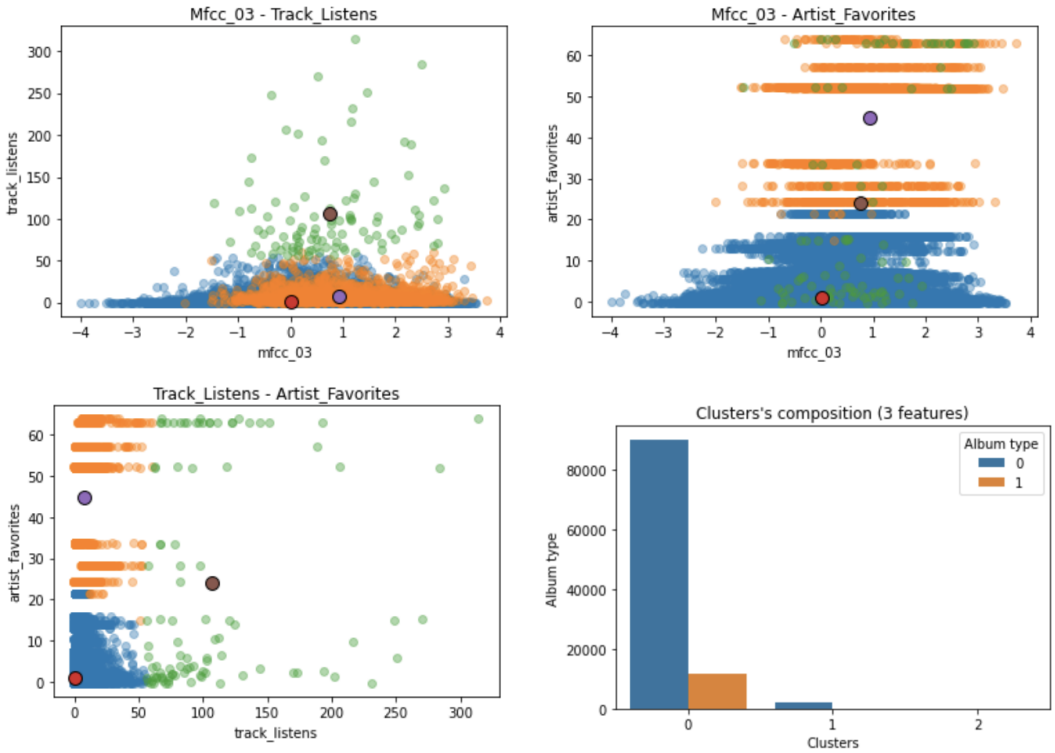


Figure 39: X-Means clustering and relative class labels distribution.

[31]The motivation behind these variables' choice is again imputed in their playing a relevant role in our previous classification tasks.
[32]Current *scikit-learn*'s implementation of K-Means only uses *euclidean distance*.

	Studio_Recording	Live_Recording	Total	Entropy	Purity
Cluster 0	90187	11525	101712	0.5098	0.8867
Cluster 1	1898	0	1898	0.0000	1.0000
Cluster 2	98	0	98	0.0000	1.0000
Total	92183	11525	103708	0.5000	0.8889

Table 25: X-Means clusters insight.

All the used variables allowed to differentiate between *cluster 0* and *cluster 1* , with *cluster 2* being an overlapping of the two. In particular, we can characterize clusters as follows:

- **cluster 0** gathers average popular tracks, displaying low artist likeness and a broad *mfcc_03* range of values;
- **cluster 1** gathers average popular tracks, interestingly belonging to the most popular artists[33] and having medium-high *mfcc_03* values;
- **cluster 2** gathers the most popular tracks, having low to high artist ratings and average *mfcc_03* values.

From the above cluster delineation, we can assert that having medium-high *mfcc_03* values usually is linked with popular tracks and/or artists, disregarding Studio and Live Recording labels.

4.2.2 OPTICS

Since all our feature combinations returned poor results, in the current report, **we retained X-Means' feature combination** for OPTICS, too, thus allowing to compare the output of the two clustering methods. For the parameter tuning, we studied *min_sample* (number of samples in a neighborhood for a point to be considered as a core point) in the [2, 50] range, using *euclidean* and *cityblock* distances. Since we got a slightly better outcome with the **cityblock** distance, we elected it as the most suitable metric to use. The best min_sample set-up resulted in **min_samples = 47** and a total of **8+1 clusters**. Clusters' performance are shown in Table 26.

N_clusters	SSE	Silhouette
8	21262.3284	0.6356

Table 26: OPTICS best results.

	Studio	Live	Total	Entropy	Purity
C -1	90170	11525	101695	0.5099	0.8867
C 0	66	0	66	0.0000	1.0000
C 1	268	0	268	0.0000	1.0000
C 2	289	0	289	0.0000	1.0000
C 3	99	0	99	0.0000	1.0000
C 4	189	0	189	0.0000	1.0000
C 5	873	0	873	0.0000	1.0000
C 6	107	0	107	0.0000	1.0000
C 7	122	0	122	0.0000	1.0000
Total	92183	11525	103708	0.5000	0.8889

Table 27: OPTICS clusters insight.

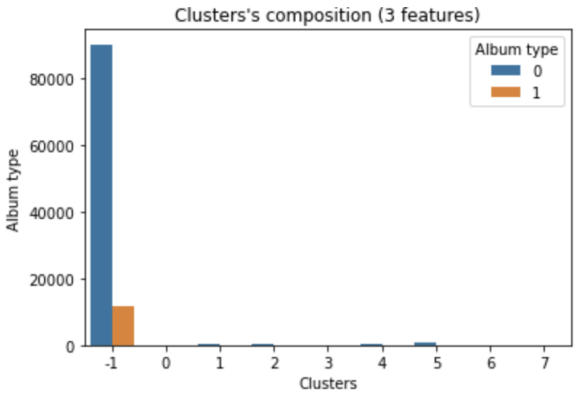


Figure 40: Studio and Live Recording distribution within OPTICS clusters.

This clustering method did not return satisfactory results since, as can be seen from the Table 27 and Figure 40, 98.06% of points were considered noisy[34], while the remaining 8 clusters contained solely Studio Recording records. From the reachability plot in Figure 41, we weren't able to detect any valley, since the noise points' one blurred them all together. This poor result has to be linked to the great similarity between our Studio and Live recordings, which makes it difficult to distinguish them. In all the previous sections, we saw how the two class labels heavily permeate, thus forming a very dense area, difficult to segment, confirmed by the current clustering task, too.

[33] An explanation to this phenomenon could be the fact that an artist becomes famous thanks to a hit, followed by average notorious songs.

[34] For OPTICS, all our Live Recordings were treated as outliers.

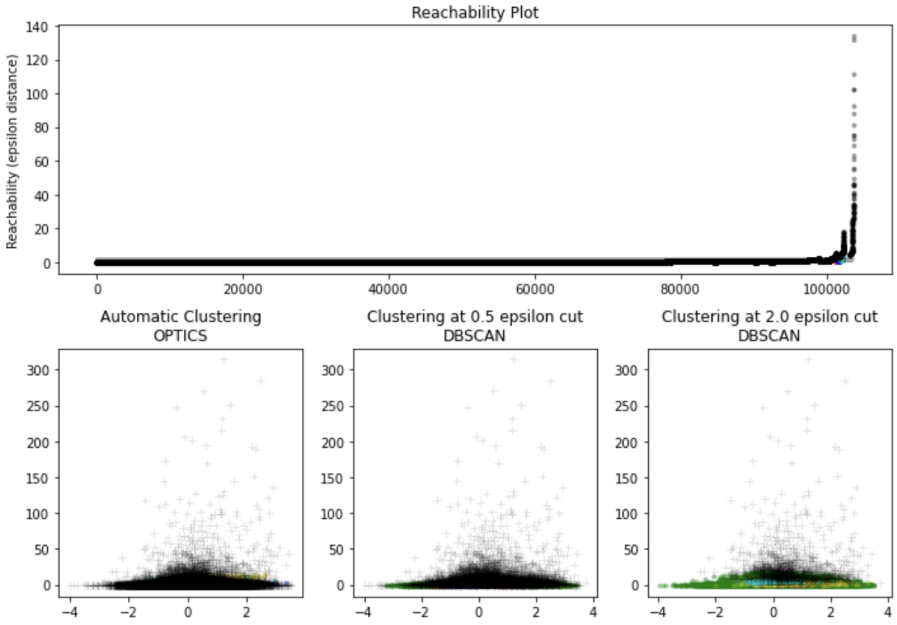


Figure 41: OPTICS reachability plot.

4.3 Transactional Clustering

As transactional clustering we ran **K-Modes** and **TX-Means**, using our categorical feature (i.e. *track_ genre_ top*) and both our ordinal one (i.e. *track_ date_ created_ year* and *track_ date_ created_ season*), to enhance the number of features. As for the advance clustering, we tried all possible feature combinations for both clustering methods. Finally, all features were codified using **OneHotEncoder**.

4.3.1 K-Modes

K-Modes' best feature combination resulted in: **track_genre_top** and **track_date_created_season**. We tuned the *n_ clusters* parameter in the [2, 20] range and the *init* parameter, evaluating between *Huang* and *Cao*. The best tuning resulted in **n_clusters** = **4** and **init** = **Cao**. The best, obtained clustering scores are shown in Table 28.

N_clusters	SSE	Silhouette
4	87681.9050	0.3732

Table 28: K-Modes best results.

The *track_ date_ created_ season* feature allowed us to distinguish 4 clusters. In particular, *Spring* allowed us to distinguish cluster 1, *Summer* cluster 3, *Autumn* cluster 2 and *Winter* cluster 0, as shown in Figure 43. This confirms what already had been detected in Section 1.2.1, thus *Winter* is usually the season in which more Studio and Live Recording creations take place. Given that the seasons have proved decisive in the clustering of records, we detected the following patterns, regarding genres: *Experimental*, *Folk*, *Classical* and *Instrumental* have a greater concentration in *Spring* and *Winter*, *Spoken* in *Fall* and *Winter*, *Old-Time/Historic* in *Summer* and *Fall*, *Pop* and *Easy Listening* in *Spring*, *Hip-pop*, *International* and *Country* in *Winter*, with the remaining genres being spread all-year around.

	Studio	Live	Total	Entropy	Purity
C 0	25546	4178	29724	0.5857	0.8594
C 1	25582	2705	28287	0.4550	0.9044
C 2	21537	1970	23507	0.4154	0.9162
C 3	19518	2672	22190	0.5306	0.8796
Total	92183	11525	103708	0.4997	0.8889

Table 29: K-Modes clusters insight.

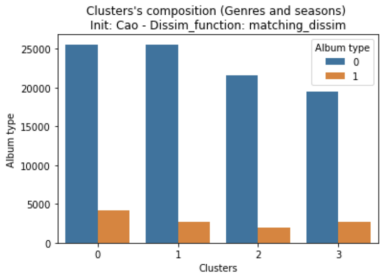


Figure 42: Studio and Live Recording distribution within K-Modes clusters.

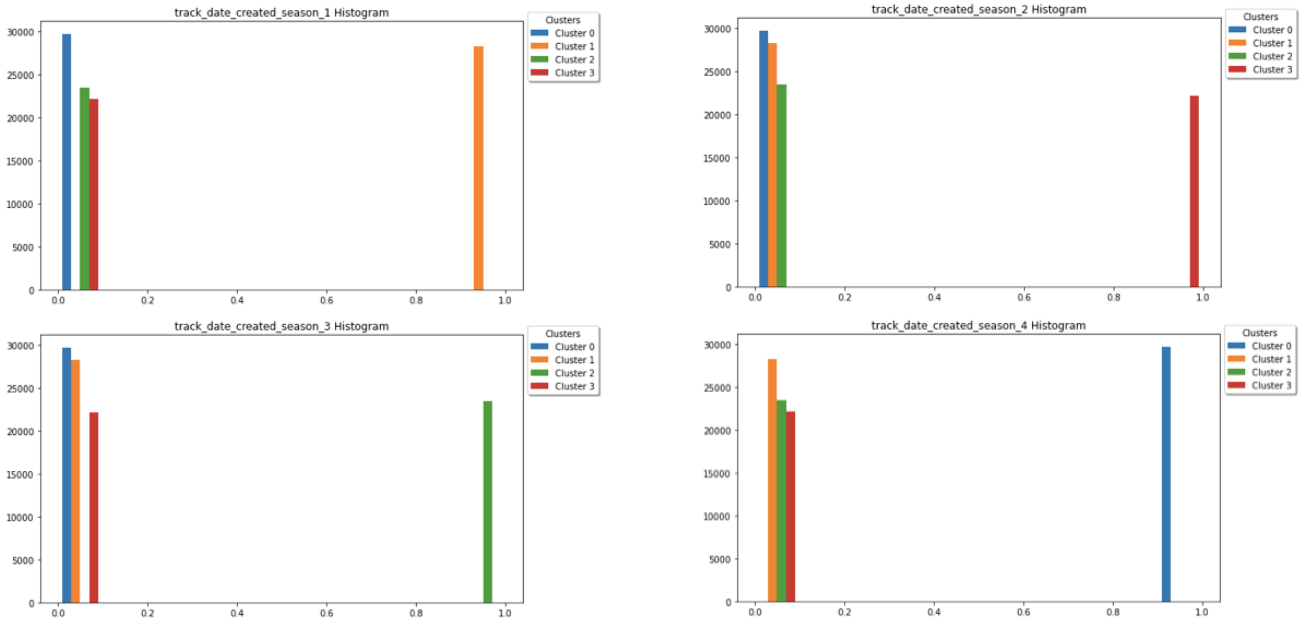


Figure 43: Distribution of seasons in K-Modes clusters.

4.3.2 TX-Means

As TX-Means is a parameter-free clustering algorithm, we tuned solely our formerly feature combinations. In Table 30, we show the best result obtained, again, using **track_date_created_season** and **track_genre_top** features. Table 31 and Figure 44 display the **4 clusters** found and additional insight in them.

N_clusters	SSE	Silhouette
4	88675.4001	0.3731

Table 30: TX-Means results.

As for K-Modes, the *track_date_created_season* feature allowed us to distinguish 4 clusters. In particular, *Spring* allowed us to distinguish cluster 1, *Summer* cluster 3, *Autumn* cluster 0 and *Winter* cluster 2, as shown in Figure 45. Again, Winter is confirm to be the most prolific season both for Studio and Live Recording labels. As for the distribution of musical genres in the seasons we can observe patterns very similar to those recorded in the clustering of K-Modes.

	Studio	Live	Total	Entropy	Purity
C 0	21537	1970	23507	0.4154	0.9162
C 1	25582	2705	28287	0.4550	0.9044
C 2	25546	4178	29724	0.5857	0.8594
C 3	19518	2672	22190	0.5306	0.8796
Total	92183	11525	103708	0.4997	0.8889

Table 31: TX-Means clusters insight.

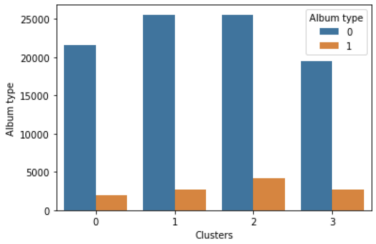


Figure 44: Studio and Live Recording distribution within TX-Means clusters.

4.4 Final evaluations

Given OPTICS' poor performance, X-Means is the best among our advanced clustering methods. For our transnational clusters, instead, we have an equity between K-Modes and TX-Means.

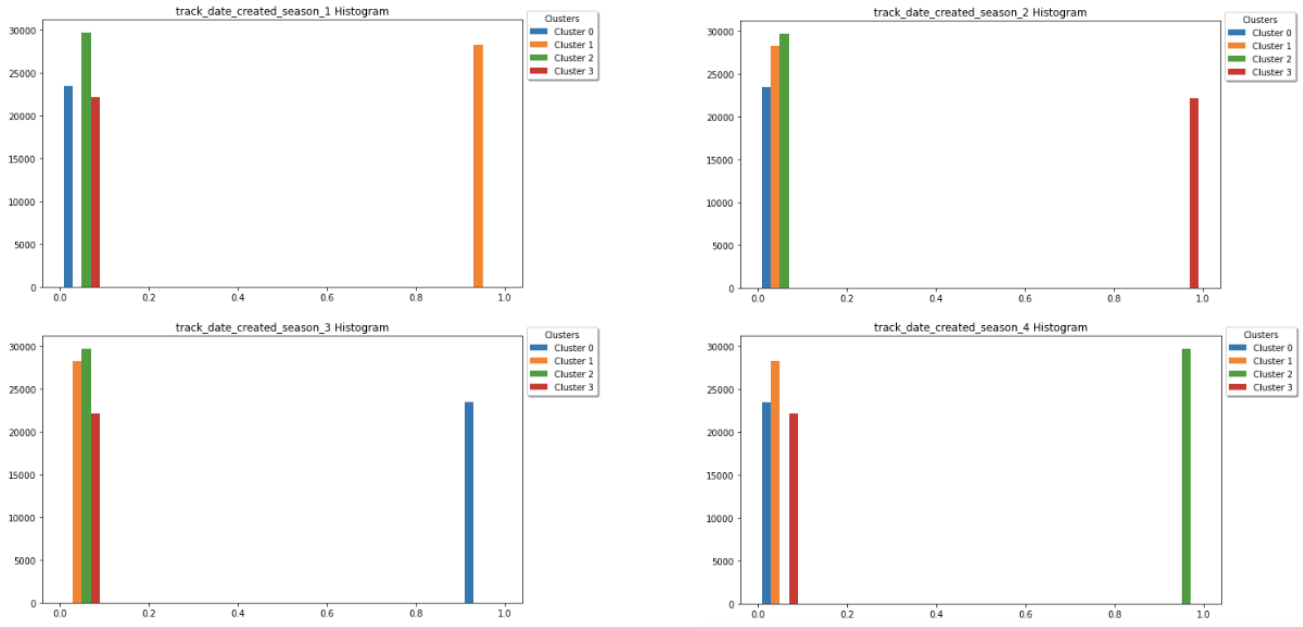


Figure 45: Distribution of seasons in TX-Means clusters.

5 Module 5

5.1 XAI (Explainable AI)

Since our best performing classifier (Section 2.1, Paragraph 2.1.4) is a Neural Network made-out of 1 hidden-layer[35], to get more insight in its prediction, we relied on the following two explainability methods:

- **SHAP**, used as a global explainer (thus, it takes into account all records);
- **LIME**, used as a local explainer (thus, it takes into account single records).

Since both approaches are computationally heavy, we focused our attention upon model's Live_Recording, correctly classified records. These helped us in better understanding what variables and relative values allowed their correct classification and as a consequences what traits weren't share with any other record (correct, Studio_Recording classification or Studio_Recording/Live_Recording misclassification).

5.1.1 SHAP (SHapley Additive exPlanations)

According to MLPClassifier's SHAP summary plots in Figure 46, Live_Recording prediction is associated with the following characteristics: 1. low *rmse_01*; 2. high *spectral_contrast_07*; 3. high *mfcc_04*; 4. high *mfcc_02*; 5. high *spectral_contrast_05*.

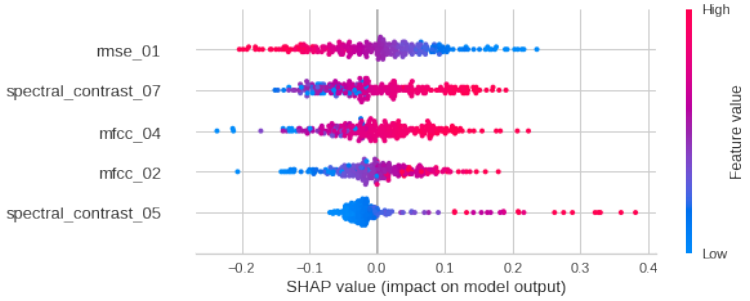


Figure 46: MLP's SHAP summary plot (top-5 features).

The most interesting distribution of effects among the top-5, SHAP contribution plot variables, is the one showed in Figure 47. It indicates how the higher *spectral_contrast_07*; *mfcc_04*, *mfcc_02* and *spectral_contrast_05* are, the higher the model's prediction is for Live_Recording. The other way around, the lower *rmse_01* is and the

[35] Consequently, it is a black-box model, for which is not clear how the classification takes place.

higher the model's prediction is for Live_Recording. It is interesting to notice how given a x-coordinate variable, *spectral_contrast_07* assumes a high value in the direction of the greatest prediction probability, too.

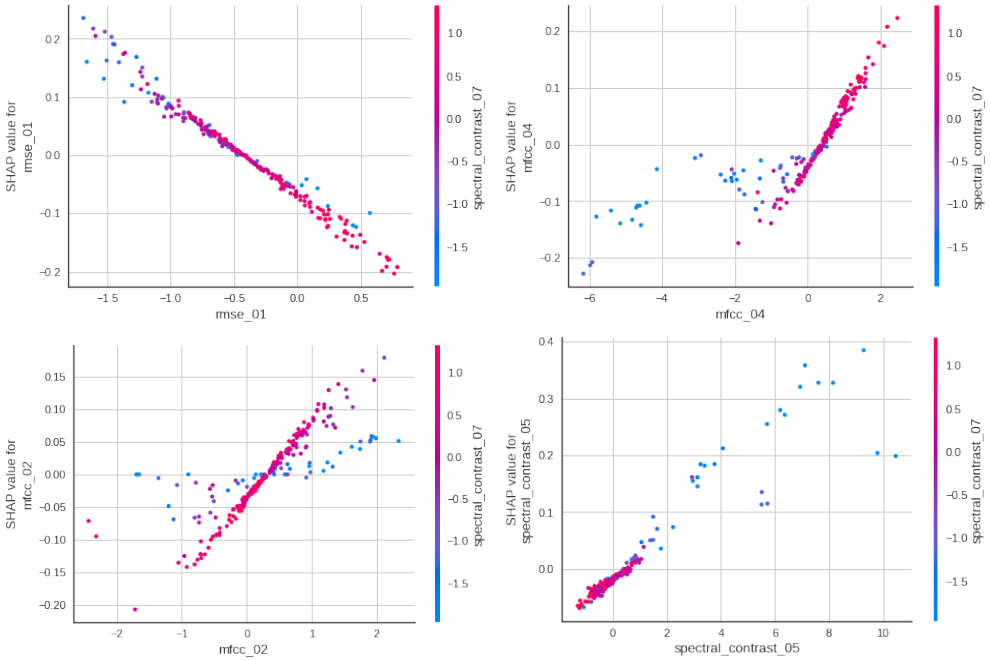


Figure 47: MLP's SHAP contribution plot (*spectral_contrast_07* vs other four, top-5 features).

5.1.2 LIME (Local Interpretable Model-Agnostic Explanations)

Using LIME to explain each individual, correctly classified, Live_Recording prediction, resulted in 211 out of 243 records displaying at least one of the features highlighted by MLPClassifier's SHAP summary plot in Figure 46[36]. Regarding MLPClassifier's SHAP contribution plot, with the help of LIME's local insight we were able to specialize features' low or high values, as shown in Figure 47. According to LIME, Live_Recording prediction is associated with the following characteristics: 1. *rmse_01* <= -0.72; 2. *spectral_contrast_07* > -0.62; 3. *mfcc_04* > -0.52; 4. mfcc_02 > 0.64; 5. *spectral_contrast_05* > 0.40.

Figure 48: MLPClassifier's LIME more certain Live_Recording record classification.

Figure 49: MLPClassifier's LIME borderline Live_Recording record classification.

Figures 48 and 49 display, respectively, the highest (*track_id*=22.199) and lowest (*track_id*=34.312) Live_Recording prediction probabilities encountered in our MLP, black-box classification analysis.

[36]The remaining 32 correctly classified, Live_Recording records displayed, instead, SHAP summary plot's following variables, such as *track_bit_rate*, *artist_favorites*, *chroma_cens_07*, *mfcc_03*, *track_listens* and *track_duration*.