SpotyPY

SAPIENZA
UNIVERSITÀ DI ROMA

Presentation of our work for the STAT4ACSAI 21-22 course’s project

Yusupha Juwara
Benjamin Barda
FrancescoDanese

Overview

- For this project we decided to dive deep into some of the many models and tools seen during the duration of the course.
- We decided that, given our shared passion for music, focusing on this field would be the perfect match of pleasure and "duty".
- During this brief presentation we present both the challenges and the result we had, in addition to future additions.
- For the full repository of this project visit our public github repository.

SAPIENZA
UNIVERSITÀ DI ROMA

The process

- We could divide the journey of this project into 3 main phases :
  - Data gathering
  - Implementation from scratch
  - Exploring the results

Data gathering

- It was always clear to us that trying to analyse the whole spectrum of existing genres was going to be too big of a task for our limited resources and we were fearing that nothing meaningful would stem out of this approach.

- For the reason mentioned above we decided to limit ourselves to four genres, that in our opinion have both similarities and differences with respect to one another :
  - Jazz
  - Metal
  - Rap
  - Classical

SAPIENZA
UNIVERSITÀ DI ROMA

Data gathering

- Once defined the scope of the project we started gathering data. Our first idea was to use the Librosa package in order to extract features from audio files. This proved to be unfeasible indeed given that downloading huge amount of audio-files, and label them, was an effort out of reach for us.

- As an alternative we used the Spotify API, which provide among many things, precomputed features for each song, making the data gathering process faster. Of course we had accepted a trade-off given the fact that the Librosa analysis is much more exhaustive than the features provided by Spotify.

Data gathering

- On top of being simpler to store and retrieve, the Spotify precomputed features are “human understandable” and a more detailed explanation is provided in the Spotify developer guide. The ones we considered were :

    - Mood related : Danceability, Valence, Energy, Tempo
    - Loudness, Speechiness, Instrumentalness
    - Context related: Liveness, Acousticness

Data gathering

- In order to build the dataset we downloaded the features appearing in the top Playlists for each one of the genres allowing us to quickly have labelled data.

- After grouping them and cleaning the data frames from missing values and duplicates, we found ourselves with around 1300 datapoints. Which we considered a size good enough to be able to process it with ease on our modest machines.

Implementation Part

- Once data has been gathered we decided that it was going to be academically beneficial to code our own models from scratch, trying to avoid as much as possible to use pre-computed modules.

- To make it a fair mix of parametric vs non-parametric, classifications vs clustering algorithm we decided to implement the following.
  - Random forests
  - K-means (++)
  - MoG (EM)
  - GNB
  - Linear and logistic regression models.

- We will not dive deep into the math of each model since this is presented both in the notebooks as well as in the report.

- We think that would be more interesting for the audience to look at the results instead and provide a comparison on the performance of the different models

- Now to the MODELs

SAPIENZA
UNIVERSITÀ DI ROMA

Decision trees

- Decision trees are well known tools used for classification as well as for regression.

- Our implementation of the decision Tree uses the Information gain as the objective function.

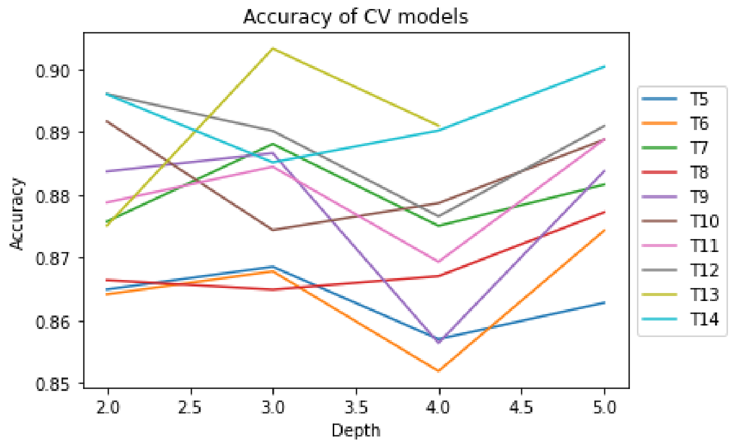
- We trained the Tree on a 75/25 split for training and evaluation giving us on average an accuracy of 86%.

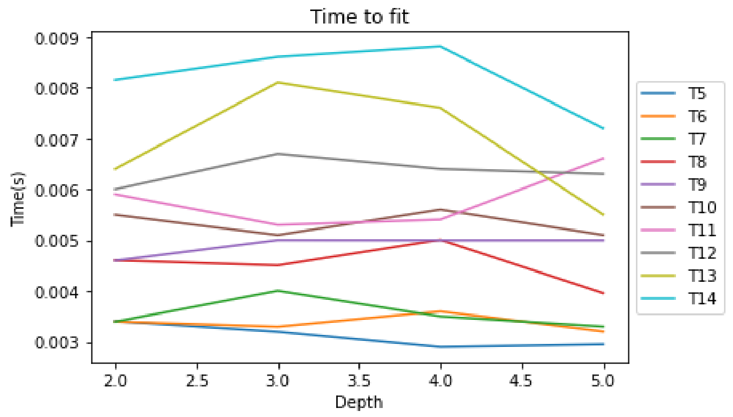
- The main purpose was for this implementation to be the base of our random forest.

SAPIENZA
UNIVERSITÀ DI ROMA

- Having implemented the random forest we decided that it was the perfect opportunity to apply some model selection techniques.

- We performed a 10-fold cross validation tuning both the maximum depth of the trees as well as their number in grid search approach.





SAPIENZA
UNIVERSITÀ DI ROMA

K-MEANS, KMEANS++

- Clustering is one of the most common exploratory data analysis techniques used to get an intuition about the structure of the data.

- It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.

- In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance.

- The decision of which similarity measure to use is application-specific.

- Clustering is used in market segmentation; where we try to find customers that are similar to each other whether in terms of behaviors or attributes, image segmentation/compression; where we try to group similar regions together, document clustering based on topics, etc.

- In our case, clustering based on music genre of either being "Classic", "Jazz", "Metal" and "Rap".

SAPIENZA
UNIVERSITÀ DI ROMA

Kmeans Algorithm

- Kmeans algorithm is an iterative algorithm that tries to partition the dataset into "K" pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible.
- It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster’s centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

- That is, K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-of-squares criterion:

- $\sum_{i=1}^{n} \min_{\{\mu_j \in C\}} (|| x_i - \mu_j ||^2)$

- Inertia can be recognized as a measure of how internally coherent clusters are

- The way kmeans algorithm works is as follows:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids; i.e assignment of data points to clusters isn't changing.
    1. Compute the sum of the squared distance between data points and all centroids.
    2. Assign each data point to the closest cluster (centroid).
    3. Compute the centroids for the clusters by taking the average of all data points that belong to each cluster.

-

- The approach k-means follows to solve the problem is called Expectation-Maximization.

1. The E-step is assigning the data points to the closest cluster.

2. The M-step is computing the centroid of each cluster.

k-means++

- This algorithm ensures a smarter initialization of the centroids and "improves" the quality of the clustering.
- Apart from initialization, the rest of the algorithm is the same as the standard K-means algorithm.
- That is K-means++ is the standard K-means algorithm coupled with a smarter initialization of the centroids.
- Still sensitive to outliers as we shall explain in a later section.

The algorithm in plain English:

1. Choose one center uniformly at random among the data points.

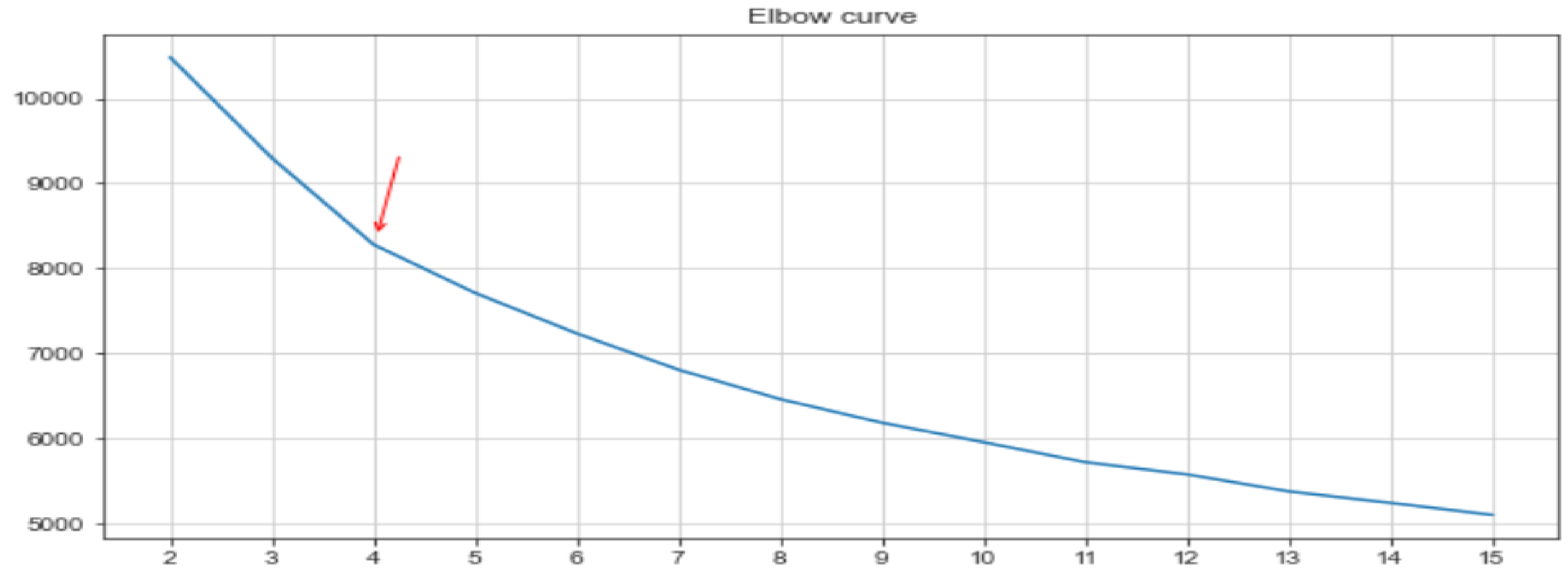
2. For each data point x not chosen yet, compute D(x), the distance between x and the nearest center that has already been chosen.
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.

4. Repeat Steps 2 and 3 until k centers have been chosen.
5. Now that the initial centers have been chosen, proceed using standard k-means clustering.

SAPIENZA
UNIVERSITÀ DI ROMA

ELBOW CURVE

- Let's see what the Elbow curve tells us about the appropriate number of clusters, we hope and expect this to be 4, because it would mean that those classes are "differentiated" and therefore our work to predict them using those features is meaningful.



- Even if it's not the most concise and strong curvature, we observe a great drop in dispersion when k = 4 with respect to 2 or 3, thus k=4 seems a fairly good choice for our dataset.

Confusion Matrix, Classification Report and Accuracy Score

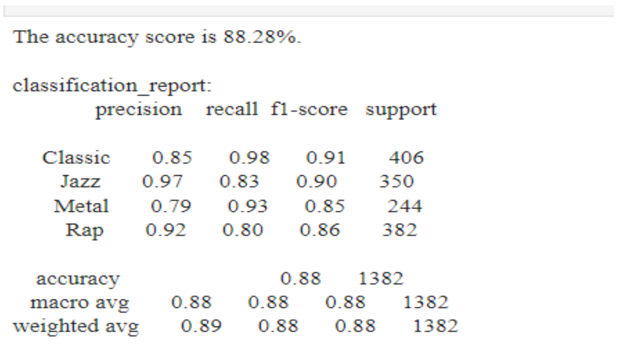
- We know that in real life scenarios we don't have the ground truth labels when dealing with unsupervised algorithm.
- But for testing purpose, we come up with a kind of accuracy score for kmeans.
- That is, after the algorithm has converged we compute the mode of the real labels in each cluster, and assign that cluster to represent that precise label, then compute the accuracy and confusion matrix according to that.

- That is, a wrapper function of <Confusion Matrix, Accuracy Score and Classification Report> in order to get the true labels match the predicted labels. After the label matching, this function calls our earlier function called confusion_matrix_score.

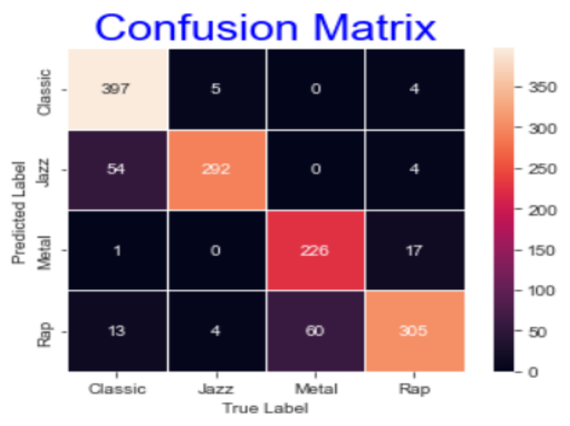
- With this, we do not need some sort of bipartite graph to get the matching done

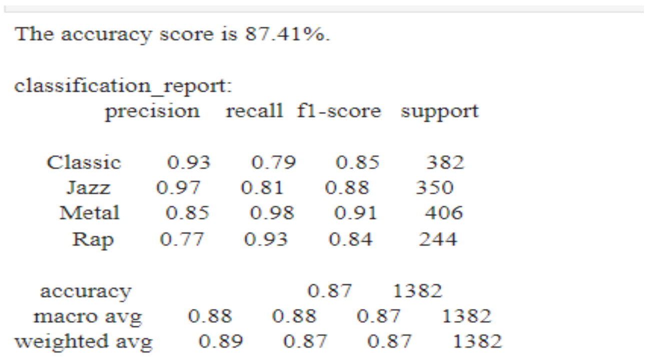
- The results are below.

Sk-learn's on the left and ours on the right

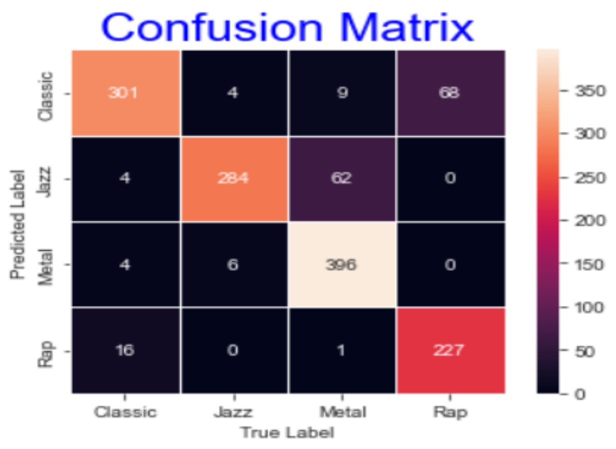


```
The accuracy score is 88.28%.

classification_report:
            precision    recall  f1-score   support

     Classic       0.85      0.98      0.91       406
        Jazz       0.97      0.83      0.90       350
       Metal       0.79      0.93      0.85       244
         Rap       0.92      0.80      0.86       382

    accuracy                           0.88      1382
   macro avg       0.88      0.88      0.88      1382
weighted avg       0.89      0.88      0.88      1382
```





```
The accuracy score is 87.41%.

classification_report:
            precision    recall  f1-score   support

     Classic       0.93      0.79      0.85       382
        Jazz       0.97      0.81      0.88       350
       Metal       0.85      0.98      0.91       406
         Rap       0.77      0.93      0.84       244

    accuracy                           0.87      1382
   macro avg       0.88      0.88      0.87      1382
weighted avg       0.89      0.87      0.87      1382
```



SAPIENZA
UNIVERSITÀ DI ROMA

Clustering performance evaluation

- Evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall of a supervised classification algorithm.

- In particular any evaluation metric should not take the absolute values of the cluster labels into account but rather if this clustering define separations of the data similar to some ground truth set of classes or satisfying some assumption such that members belong to the same class are more similar than members of different classes according to some similarity metric.

- How can one measure clustering goodness of fit?

- Supervised algorithms have lots of metrics to check their goodness of fit like accuracy, r-square value, sensitivity, specificity etc. but what can we calculate to measure the accuracy or goodness of our clustering technique?

- The answer to this question is Silhouette Coefficient or Silhouette score.

- We can also use the silhouette score to check the optimal number of clusters.

SAPIENZA
UNIVERSITÀ DI ROMA

Silhouette Coefficient

- Silhouette score for a set of sample data points is used to measure how dense and well-separated the clusters are.

- If the ground truth labels are not known, evaluation must be performed using the model itself. The Silhouette Coefficient is an example of such an evaluation, where a higher Silhouette Coefficient score relates to a model with better defined clusters.

- The Silhouette Coefficient is defined for each sample and is composed of two scores:

- a: The mean distance between a sample and all other points in the same class.
- b: The mean distance between a sample and all other points in the next nearest cluster.

- The Silhouette Coefficient, s, for a single sample is then given as:

- $s = \dfrac{b - a}{\max(a, b)}$

- The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample.

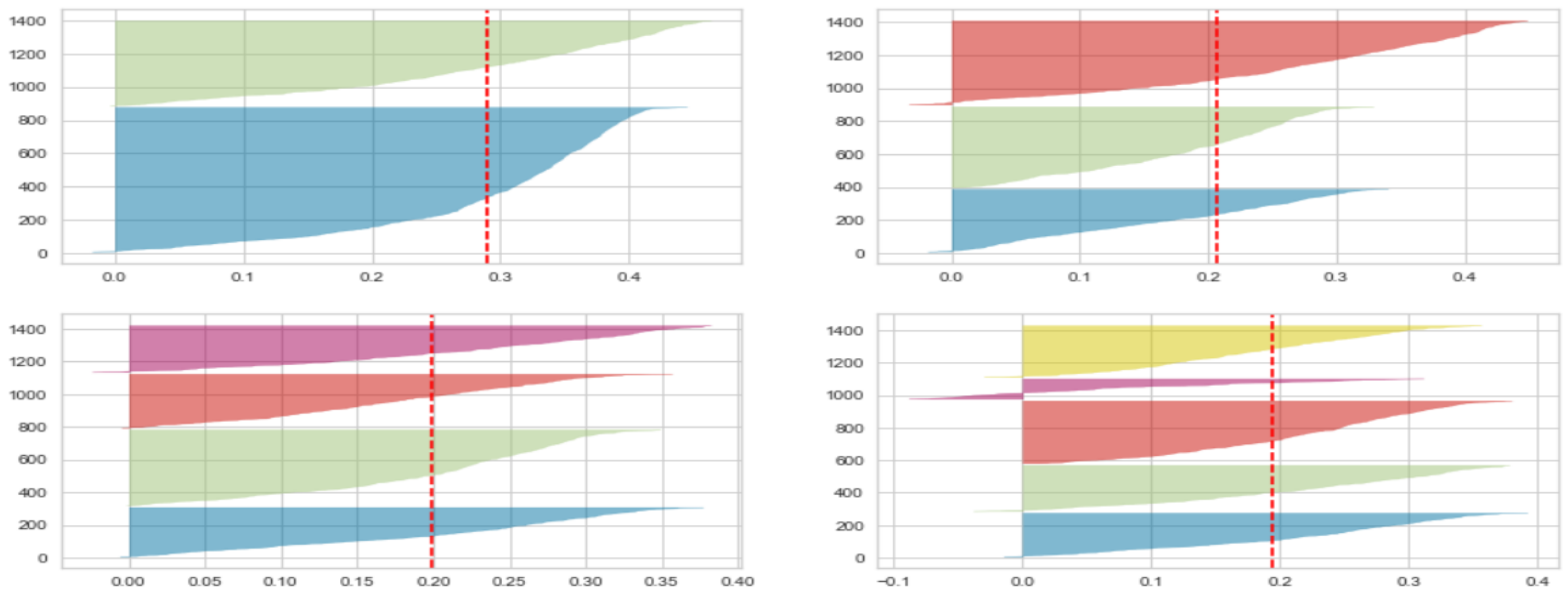
- Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.
- With our this model, the silhouette score is 0.22.
- Which implies that the samples are separated but might not be far away from their nearest neighboring clusters.

- Note that Silhouette Coefficient is only defined if number of labels is 2 <= n_labels <= n_samples - 1.

- The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

SAPIENZA
UNIVERSITÀ DI ROMA

Silhouette Score's Plot



- K =2 in top left, k=3 in top right, k=4 in bottom left and k=5 in bottom right

SAPIENZA
UNIVERSITÀ DI ROMA

Silhouette Plot Analysis

- We proceed with a quick Silhouette analysis done on the above plots to select an optimal value for the number of clusters.

- The number of clusters 2 and 5 look non-optimal for the given data due to the following reasons:

  - A large presence of samples with negative Silhouette Score (for k = 5)
  - Wide fluctuations in the size of the silhouette plots (for k = 2 and k = 5)

- In conclusion 3 and 4 clusters are a better choice in our opinion, and we are glad for it since our dataset is actually divided into 4 categories.

SAPIENZA
UNIVERSITÀ DI ROMA

Comparison between kmeans'

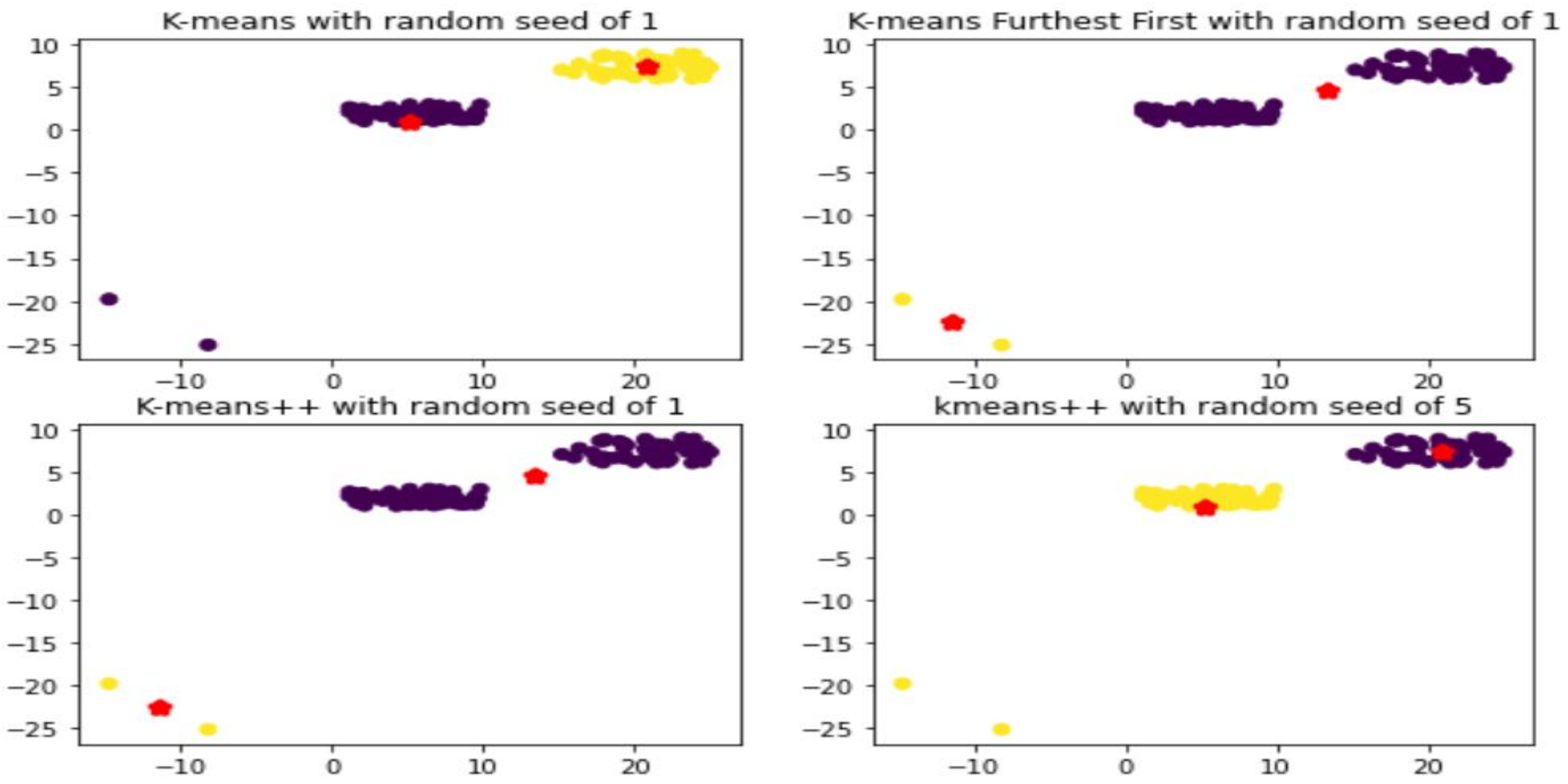
- We do not seek to compare kmeans and kmeans++ because, unlike kmeans furthest first, they are both entirely random.
- After randomly choosing first centroid, kmeans++ chooses the other k-1 centroids by giving more weight to far away points than nearby ones, which makes it more sensitive to outliers than the general kmeans.
- However, since it was developed as an improvement over kmeans, it does often a better clustering than kmeans.

- That is, far away points get large probabilities, but the probabilities of the nearby ones summed together may be large. So the sample can take any according to their proportions.

- One may wonder whether or not kmeans furthest first always does a better job in clustering than kmeans. The answer is that kmeans furthest first does not always outperform kmeans.

- A plot would help to better understand. So let's see some plots below.

SAPIENZA
UNIVERSITÀ DI ROMA



- From this toy example, when k=2, kmeans clusters the data better.
- However, for other k values, either Kmeans Furthest First outperforms it or they are on a par with each other.
- Therefore, We can conclude that Kmeans Furthest First is sensitive to outliers and does not always outperform kmeans.

Mixture of Gaussians

- A GMM is a parametric probability density function rapresented as a weighted sum of its k components.

- That is, a Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

- If we assume to have J subgroups in our dataset D, each one being **Heterogenous across** and **homogeneous within**, then each subgroup will have the same parametric family differing by the values of the parameters across the groups, then by estimating $\Theta = (\pi_1, \cdots, \pi_k, \theta_1, \cdots, \theta_k)$, we will be able to cluster unobserved realizations of X ={X_1,...., X_n} of which D={x_1,..., x_n} is a specific realization.

- In this instance we assume that we are dealing with multivariate gaussian distributions.

EM Algorithm of GMM

- So to the question on how to estimate the parameters, trying to estimate the parameters is NP given that the Log-likelihood is not jointly concave w.r.t- the mixture parameters.

- Luckily we can use the Expectation Maximization Algorithm for finding the maximum likelihood estimates of a mixture (of gaussians in our case) model.

- As the name suggests ,the algorithm is divided in two steps.

SAPIENZA
UNIVERSITÀ DI ROMA

- **Assumptions**

✓ A dataset D described as above where ∀x ∈ D, x ∈ Rd.

✓ Independence

- Moreover we define the membership weight
- $w_{\{i,j\}} = \frac{P_k(x_i \mid \theta_k)\, \pi_k}{\sum_{\{m=1\}}^{K} P_m(x_i \mid \theta_m)\, \pi_m}$
- **E-step**
- Calculate the membership weights $w_{\{i,j\}}$ for each datapoint x_i, 1≤i≤N and all mixture components 1≤k≤K. This will yield a N × K matrix.

- **M-step**
- **Let** $N_k = \sum_{\{i=1\}}^{N} w_{i,j}$ i.e. the effective number of datapoints assigned to component K.
- We now update Θ as follows:

- $\pi_k^{new} = \frac{N_k}{N} \qquad 1 \leq k \leq K$
- $\mu_k^{new} = \left(\frac{1}{N_k}\right) \sum_{\{i=1\}}^{N} w_{i,k}\, x_i \qquad 1 \leq k \leq K$
- $\Sigma_k^{new} = \left(\frac{1}{N_k}\right) \sum_{\{i=1\}}^{N} w_{i,k}\, (x_i - \mu_k^{new})(x_i - \mu_k^{new})^T \qquad 1 \leq k \leq K$

SAPIENZA
UNIVERSITÀ DI ROMA

Convergence

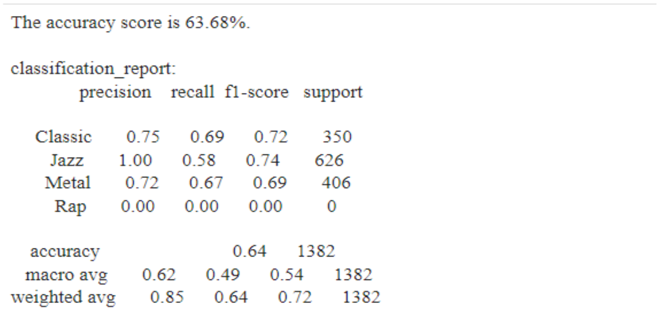
- The algorithm takes an iterative approach where in each iteration the M-step is ran after the E-step. But how do we decide when to stop the iterations?

- One approach is to limit the amount of iterations a priori.
- A better approach is to calculate at each iteration the value of the log likelihood and stop the algorithm when no appreaciable improvement is gained w.r.t to the previous iteration.
- Note that EM does not guarantee to converge neither to a global or local minima.

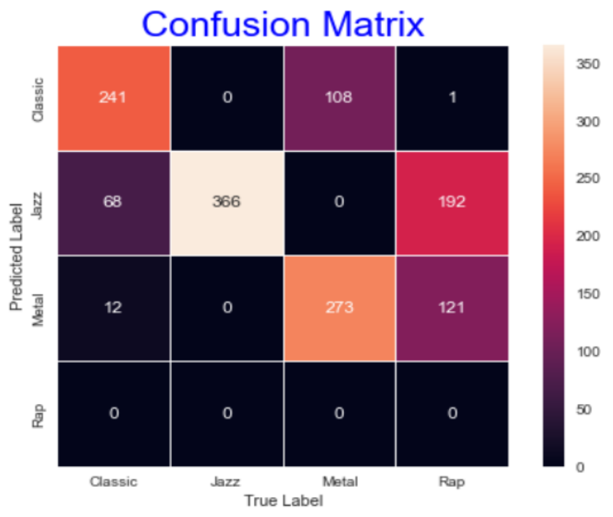
SAPIENZA
UNIVERSITÀ DI ROMA

GMM Confusion Matrix, Classification Report and Score

- With this dataset, kmeans does surely a better work with respect to GMM.
- Those unsupervised methods gave us some interesting outcomes, but now it's time to go back to our original line of thought and try to predict the class having a labelled dateset. Let's discover what the world of **Supervised Models** can achieve



```
The accuracy score is 63.68%.

classification_report:
              precision    recall  f1-score   support

     Classic       0.75      0.69      0.72       350
        Jazz       1.00      0.58      0.74       626
       Metal       0.72      0.67      0.69       406
         Rap       0.00      0.00      0.00         0

    accuracy                           0.64      1382
   macro avg       0.62      0.49      0.54      1382
weighted avg       0.85      0.64      0.72      1382
```



Naive Bayes Classifier

- Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

- There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

- For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter.
- A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

- Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $x=(x1,\ldots,xn)$ representing some n features (independent variables), it assigns to this instance probabilities P(C_k | x1, …, xn)
- for each of K possible outcomes or classes C_k

- In plain English, using Bayesian probability terminology equation can be written as
- Posterior = $\frac{prior \times likelihood}{evidence}$
- In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant.
- So the evidence is a scaling factor dependent only on x_1, …, x_n , that is, a constant if the values of the feature variables are known.

- Again, in our case, since we have four classes each of which occuring with equal probability of ¼, we can see that the **prior** is effectively constant just like the evidence above

- So we have only the **likelihood** to work with.

- Now the **naive** conditional independence assumptions come into play: assume that all features in x are mutually independent, conditional on the category C_k.

- Thus, the joint model, after dropping the prior and the evidence, can be expressed as **proportional** to $P(x_1 \mid C_k) \times \cdots \times P(x_n \mid C_k)$

- Which Distribution should we use for the joint model?.

- Even though there are many distributions to use here, we opt for Gaussian Naive Bayes.

SAPIENZA
UNIVERSITÀ DI ROMA

Gaussian Naive Bayes

- When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.

- For example, suppose the training data contains a continuous attribute, x.

- The data is first **segmented** by the class, and then the **mean** and **variance** of x is computed in each class.

- Let μ_k be the mean of the values in x associated with class C_k, and let σ_k^2 be the unbiased sample variance of the values in x associated with class C_k (that is, the degree of freedom is 1 => n-1).

- Suppose one has collected some observation value v.

- Then, the probability density of v given a class Ck, p(x=v| C_k), can be computed by plugging v into the equation for a normal distribution parameterized by μ_k and σ_k^2

- In order to classify samples, one has to determine which posterior is greater: classic, jazz, metal or rap.

- We use the **negative log-likelihood** since the product of a large number of small probabilities can easily underflow the numerical precision of the computer.

- This is resolved by computing instead the sum of the negative log probabilities.

- The best estimate for the mean and variance parameters of a Gaussian are simply the empirical estimates of the mean and variance respectively.

- These means and variances are for each feature w.r.t each class.

- For example, the feature danceability w.r.t rap has a mean and variance different than danceability w.r.t classic.

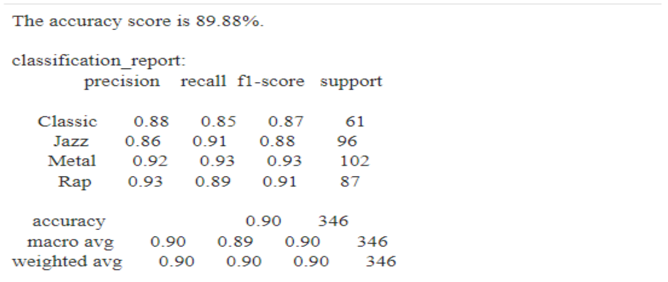
- So for each feature x, it has a mean and variance for each class c.

- Therefore, the means and variances must be calculated for each feature (in our case, a whopping 12 features for 4 classes).

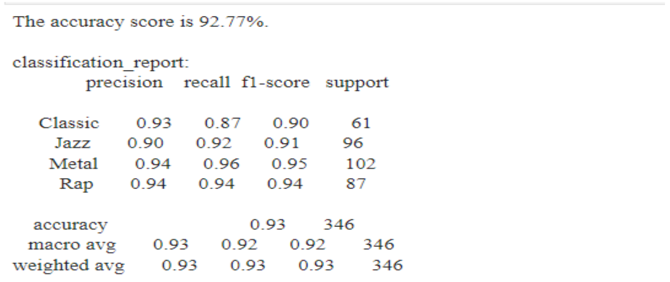
- We did this in a simple one-pass by first filtering out the appropriate instances (w.r.t to the class in question) under each feature, then take the mean and variance as can be seen in the implementation.

SAPIENZA
UNIVERSITÀ DI ROMA

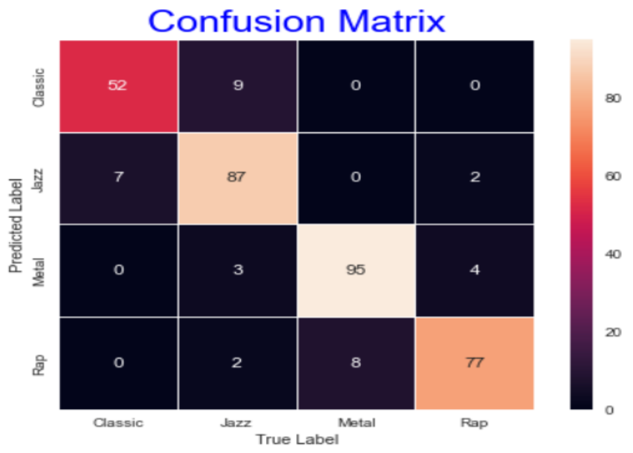
Confusion Matrix, Classification Report and Accuracy Score



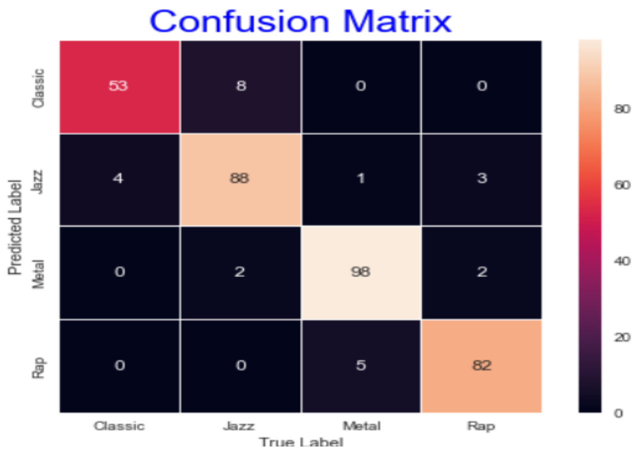
```
The accuracy score is 89.88%.

classification_report:
              precision    recall  f1-score   support

     Classic       0.88      0.85      0.87        61
        Jazz       0.86      0.91      0.88        96
       Metal       0.92      0.93      0.93       102
         Rap       0.93      0.89      0.91        87

    accuracy                           0.90       346
   macro avg       0.90      0.89      0.90       346
weighted avg       0.90      0.90      0.90       346
```



```
The accuracy score is 92.77%.

classification_report:
              precision    recall  f1-score   support

     Classic       0.93      0.87      0.90        61
        Jazz       0.90      0.92      0.91        96
       Metal       0.94      0.96      0.95       102
         Rap       0.94      0.94      0.94        87

    accuracy                           0.93       346
   macro avg       0.93      0.92      0.92       346
weighted avg       0.93      0.93      0.93       346
```



Predicted \ True	Classic	Jazz	Metal	Rap
Classic	52	9	0	0
Jazz	7	87	0	2
Metal	0	3	95	4
Rap	0	2	8	77



Predicted \ True	Classic	Jazz	Metal	Rap
Classic	53	8	0	0
Jazz	4	88	1	3
Metal	0	2	98	2
Rap	0	0	5	82

Ours on the left and sk-learn's on the right

Popularity of a song

- The Spotify for Developers API provide also another type of data: a track's popularity.

- Using the track "id" we can retrieve, with a little code, the actual popularity value for that particular track, and add these data into our dataset.
- But how do you quantify the popularity of a song?
- According to Spotify, “The popularity of a track is a value between 0 and 100, with 100 being the most popular.

- The popularity is calculated by an algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are.

- Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past."

- We think this value is also used for music recommendation and to build the various "Trend" playlists.

- So maybe it would be cool to look how things change if we create a dataset with very popular songs, to see what they have in common or if there is any pattern.

SAPIENZA
UNIVERSITÀ DI ROMA

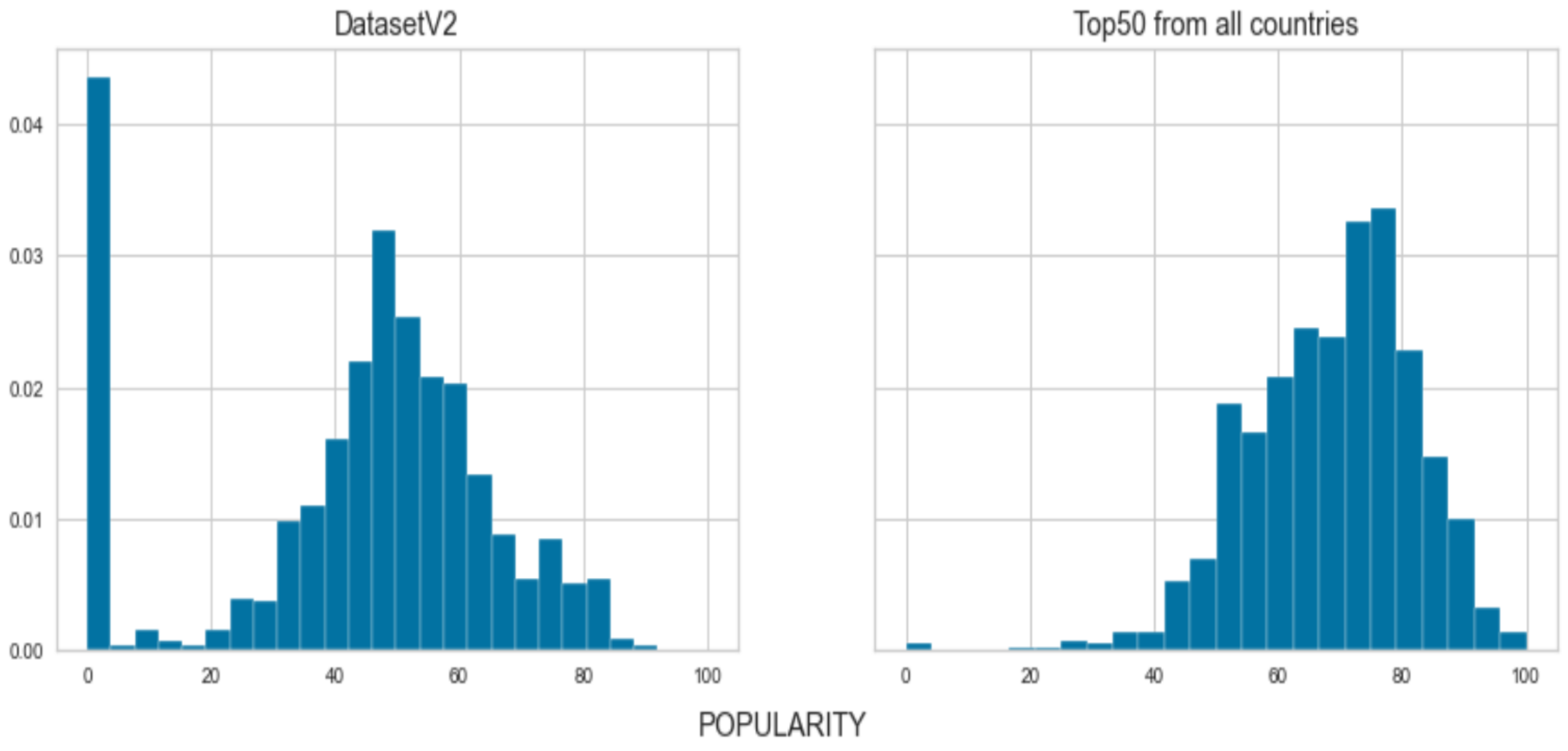
- The "Top50_clean.csv" dataset contains a set of the currently Top50 most popular songs from any country we have found on spotify. Basically we merged together the data from "Top50 Italia", "Top50 France" "Top50 Australia" and so on and so forth.

- Therefore now we have 2 datasets, the one with genres label, and the Top50 one. Let's now explore a few plots and visualizations.

- We are curious to see what the popularity distribution is across the dataset.

SAPIENZA
UNIVERSITÀ DI ROMA

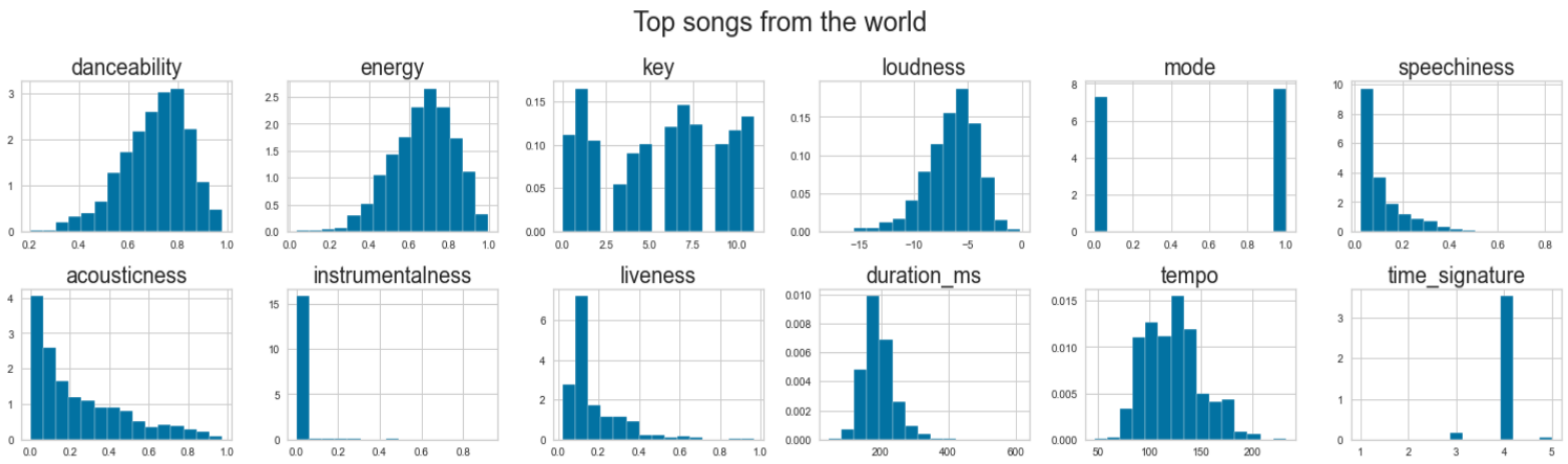
DatasetV2: mean = 43 , mode = 0
Top50_all_countries: mean = 69 , mode = 73



- If we look at the leftmost histogram, the one about the almost "popularity unbiased" dataset, we see a right skewed distribution showing us how truly rare it is to have popular song.

- Looking at the right plot, with no surprise since we took the trending songs of various countries, we see a distribution centered near high values of popularity.

- Let's now have a look to some statistics of about popular songs:



```
danceability        7.071826e-01
energy              6.689399e-01
loudness           -6.417846e+00
mode                5.162338e-01
speechiness         1.125265e-01
acousticness        2.493612e-01
instrumentalness    1.260278e-02
liveness            1.780320e-01
valence             5.677584e-01
tempo               1.233424e+02
duration_ms         1.928392e+02
popularity          6.870292e+01
artist_followers    4.688854e+06
```

SAPIENZA
UNIVERSITÀ DI ROMA

- Since we have a little background knowledge about music, we have noticed a few interesting things :

- The most common duration among trending songs is more or less 3 minute and 30 seconds, that matches with the usual Radio's songs duration. From internet: "In general, a radio-ready song is one that is three minutes, give or take 30 seconds". The mean is indeed around 3 minutes.

- We have 3 modes for the Tempo: 95 bpm, 120 bmp and 140 bpm. Those are indeed the most used Tempo in Western music. If we want to make a (statistically insignificant) example, "Shape of you" by Ed Sheeran, "Sweet Child of Mine" by Guns N' Roses and "Beat It" by Michael Jackson are played with 95,120 and 140 bpm respectively, according to a bunch of websites.

-

SAPIENZA
UNIVERSITÀ DI ROMA

- A very common time signature is 4/4, which in music theory is indeed the basic and default time signature, also the easiest one to play and follows instrumentically.

- People like high-energy songs, better if they can dance on it (see danceability and energy mean), while they don't give much attention to vocal-free and acoustic pieces (see acousticness and instrumentalness)

- Minor tone tracks and Major tone tracks don't show any significance difference in term of popularity.

SAPIENZA
UNIVERSITÀ DI ROMA

- A fair question now is: is there any connection between our features and the popolarity of a song?

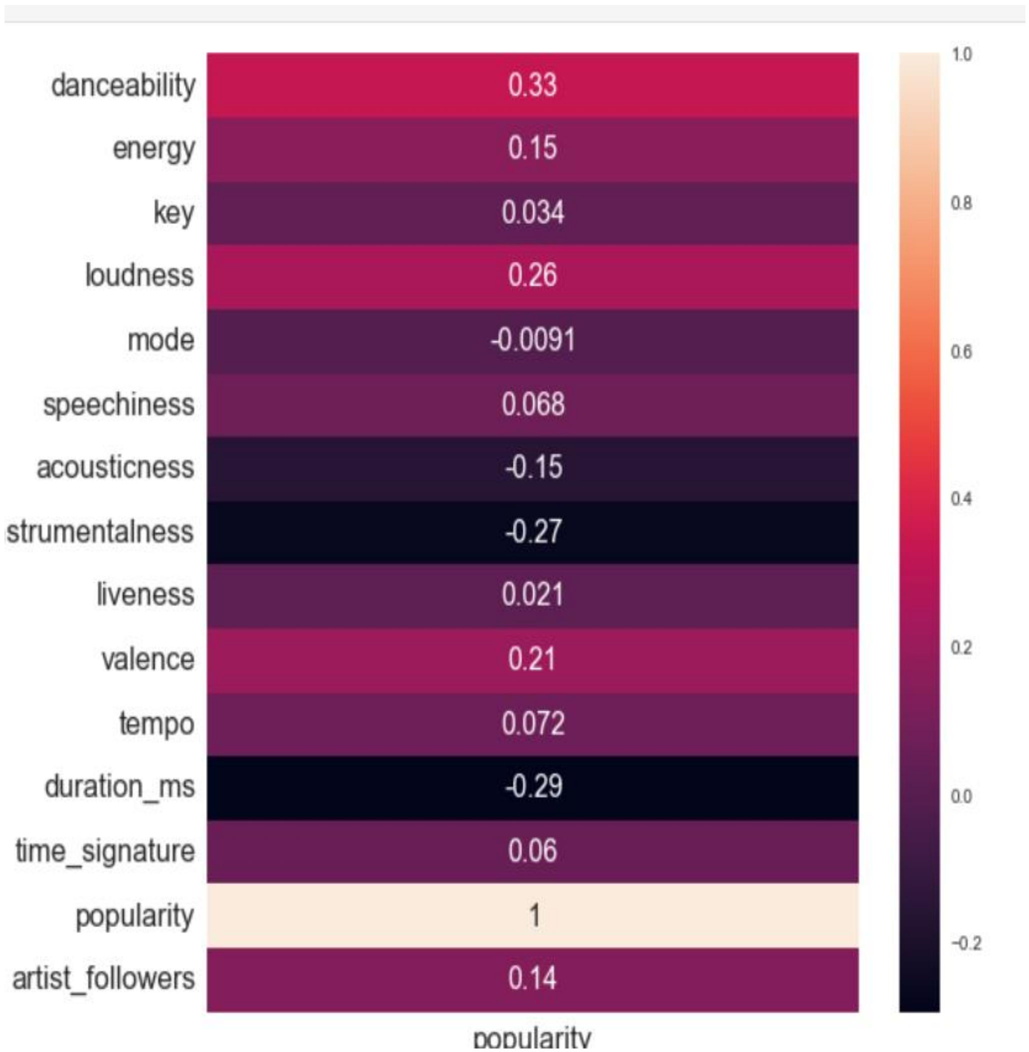
- Well, as initial guess (spoiler alert) we don't expect too much dependeces since how much a track will be famous depends a lot about external factors, as the current fashion, events happening in the world, artist's importance and reputation, social-media stuff and advertisment and so on.

- But let's see if we can come up with something notable.

SAPIENZA
UNIVERSITÀ DI ROMA

- First of all, let’s look at a correlation table to identify some baseline correlations between our many X variables.

- To do so we merge our two datasets onto one and see the results.



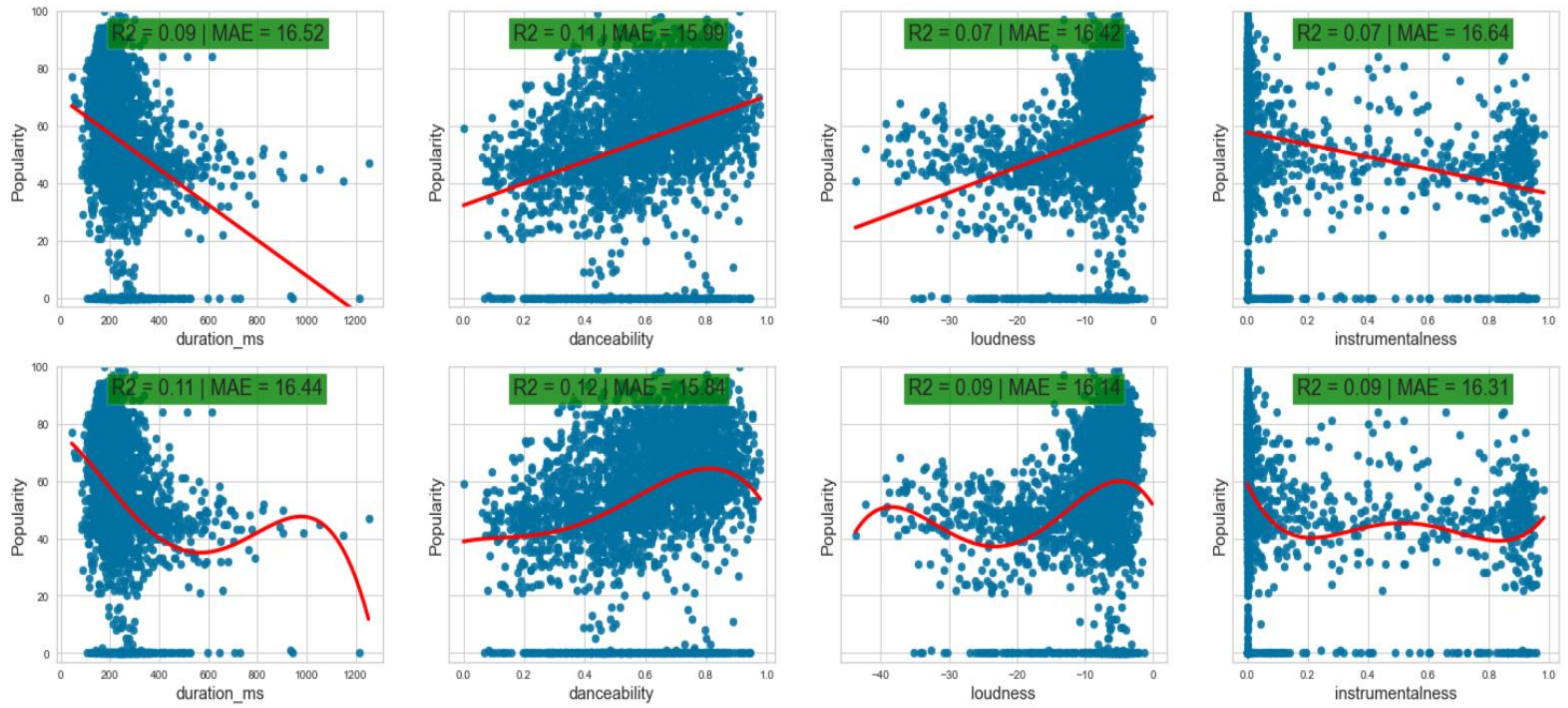
- As expected, we can't see any strong correlation. The best results come with *danceability*, *loudness*, *instrumentalness* and *duration*. The latter is anticorrelated with a coefficient of -0.29.

- Let's try to plot those features and see what a simple Linear Regression model looks like.

- We'll use our Linear Regression class, implemented by scratch, to see how a line visually fits the data. The first row of subplots shows a classic linear regression, the second a polynomial regression of 4th degree. We then output the R2 coefficient and Mean Absolute Error of each fit.

SAPIENZA
UNIVERSITÀ DI ROMA

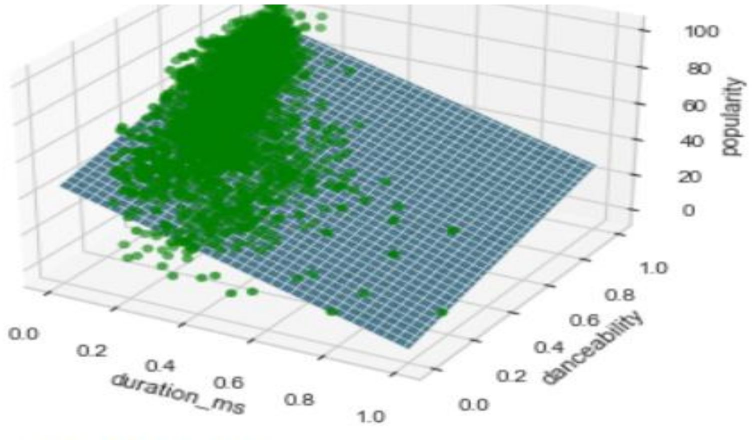
Regresion Plots of Popularity



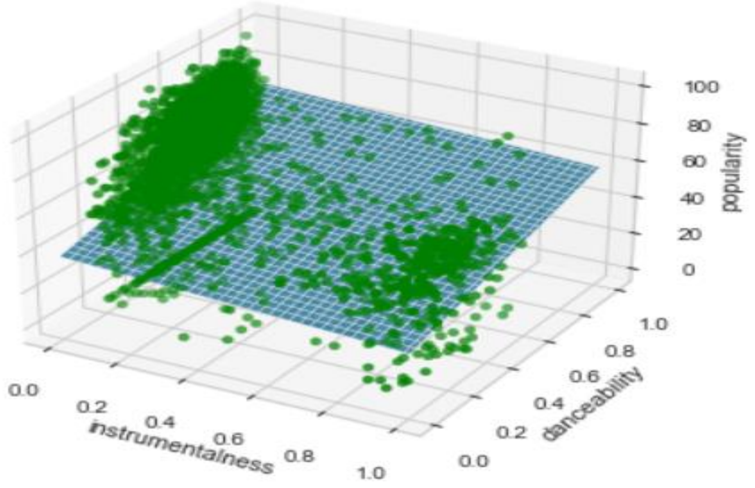
- The best result is the Polynomial Regression of 4th degree using the **danceability** of the track as explanatory variable with an R-squared coefficient of 0.12 and A Mean Absolute Error of 15.84.

- An average of 15.84 spread for our residuals in the prediction model for a range of 0–100 in ‘popularity’ is huge. And also R2 is low.

- Let's try adding a dimension.



R2 = 0.14 | MAE = 15.8



R2 = 0.12 | MAE = 15.91

- Things do not really seem to get better, our goal feature is probably not really linear correlated to the others.

- So perhaps it’s time to see what the classification world could offer us.

- **Classification**
- In order to set up any classification model we have to move away from trying to predict a continuous integer value, and instead modelling classes/label.

- So an idea could be to divide the 0 - 100 range of *"popularity"* into 3 sorted sections.

- We created three bins that represent "low","medium","high" popularity using pandas.cut() and then split the dataset into training and test set with the help of Sklearn.

- The first method we are willing to try is classification using a Decision Tree and of course a Random Forest to see how an ensemble method can perform in this scenario.

- We used our python classes built from scratch and see what this models offered us with.

- A single deterministic Decision Tree pops out an accuracy score of about 0.65.
- So with the model we have a probability to get the label right that is twice as much as randomly choosing among 3 classes (33% if we casually pick one).

- But it's still not great, we would prefer something around 80-90%, because we have to remember that we have created only 3 categories over a continous range of length 100.

- For instance if we divide instead the dataset into 5,10,20 bins we'll see the accuracy dropping very fast.

- With the Random Forest we don't see any major improvement, but since this method relies on *Bootstrapping* the result can change a bit over different code runs/random seeds.

SAPIENZA
UNIVERSITÀ DI ROMA

Conclusion part

- Our (and your) journey here is finally coming to an end.

- We really had fun doing all of this statistical analysis and Data Science.
- We have found nice curiosities about songs, genres, popularity and we have observed how different models and methods performs with this kind of real-world dataset.

- But leaving back all the accademical purposes, we also think that a work of this type can be somehow adapted and used to build reccomendation systems, targeted advertising, trend and fashion analysis and other possible useful applications that a company can come out with

- Also we are conscious that a huge amount of data are private and kept hiden by Spotify, since we have to remember it is still a multimillionarie dollars company.

Sum up below

Quick Sum-up

- Our work dealt with the folowing:

- Data statistics and visualizations: features

✓ mean,

✓ mode,

✓ correlations,

✓ scatterplot and histograms,

✓ genres pecularities and similarities.

- Clustering Algorithms:

✓ Kmeans with different initializations such random, ++ , furthest first;

✓ Gaussian Mixture models;

✓ Evaluation metrics such as Elbow method, silhouette score and plots;

- Classification models to predict the genre:

✓ Gaussian Naïve Bayes,

✓ Decision Trees and Random Forest

- Regression and classification models to predict the popularity:
  - ✓ Linear/polynomial/multiple regression,
  - ✓ Decision Trees and Random Forest,
  - ✓ Logistic regression

- It's been a fun and productive experience and we can't stop wondering how many other exciting studies we could do with extra data and time.
- We hope it was also a pleasure for the readers to take a look at our elaborate.

Francesco Danese

Yusupha Juwara

Benjamin Barda

Thank you all for your kind attention

SAPIENZA
UNIVERSITÀ DI ROMA