Maria Giustiniano
Simone Quadrelli

Song clustering

Academic year 2019 - 2020

Contents

We declare that this material,which we now submit for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work. We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.

Introduction

The project addresses the challenge of clustering songs of different genres contained in gtzan genre collection dataset. It explores the usage and performance of hierarchical and agglomerative clustering techniques while mastering the computational complexity of the proposed solutions.

We implemented a feature extraction function that operates directly on audio files to produce their numerical representation. We designed an analytics pipeline to scale up efficiently with data. We also employ plots and images to enrich our analysis and to assess the validity of the results.

Overall, the clustering techniques seem to be capable of correctly identify some genres such as metal, classic and reggae. However, other genres do not belong to defined clusters but form a big agglomerate. Consequently, some clusters have not a straightforward interpretation and may not be associated with a genre.

We divided the report into five sections whose content is reported below.

In section 1, we summarize the dataset analyzed and the features we have extracted.
Section 2 concerns the methodology and contains the description and analysis of the algorithms applied.
In section 3, we provide an overview of the processing pipeline and experiments. Moreover, we discuss the choices concerning preprocessing and dimensionality reduction techniques.
Section 4 reports the results of the analysis that we discuss in-depth in section 5.

1 Dataset and features

1.1 Dataset

We analyzed a musical dataset, called GTZAN genre collection, containing a thousand wav audio files.

It was collected between 2000 and 2001 by George Tzanetakis who decided to include songs from different audio sources (CDs, radios and microphone recordings). This variety of sources introduce some noise but helps the analysis to be more resilient in practical applications [1].

The dataset does not provide the entire songs. Instead, the 30-seconds long tracks encode the intro of songs belonging to ten different genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock.

1.2 Features

To carry out feature extraction, the audio files are divided into m frames, each one of length L. $x_i(n)$ represents a sample, where $n = 1, ..., L$ is the length of the frame and i is its index.

In the following we examine the feature extracted.

The Zero-Crossing Rate (ZCR) of an audio frame is the rate of sign-changes per frame [2]. It can be computed as:

$$zrc(i) = \frac{1}{2L} \sum_{n=1}^{L} |sgn(x_i(n)) - sgn(x_i(n-1))|. \tag{1}$$

It captures percussive sounds since the are characterized by a spike in the signal followed by a deep valley.

Chroma_sftf stands for chronogram short-time Fourier transform. Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave. Since, in music, notes exactly one octave apart are perceived as particularly similar, knowing the distribution of chroma, even without the absolute frequency, can give useful musical information.

The root-mean-square (rms) of a signal is another relevant feature. It describes the average power of a signal. Since the analysis conerns musical domain,

rms is the loudness expressed in decibels (dB). It can be defined as:

$$rms(i) = \sqrt{\frac{1}{L}\sum_{n=1}^{L} x_i(n)^2} \tag{2}$$

Mel-frequency cepstral coefficients (mfcc) represents the rate of change in spectral bands [2]. Even if there exist other cepstral coefficients, the mel implementation is particularly interesting since it relates the perceived frequency of a tone to the measured frequency. Consider a sound with frequency f in Hz, then its mel value can be expressed as:

$$mel(f) = 2595 \log\left(1 + \frac{f}{700}\right). \tag{3}$$

From each extracted frame, a procedure computes the mfcc features. First, it applies the fast Fourier transform to the frames, then, converts them in mel scale. Finally, it computes a fast Fourier transform of the logarithmically scaled mel values.
We provide a brief overview of the procedure in Figure 1.

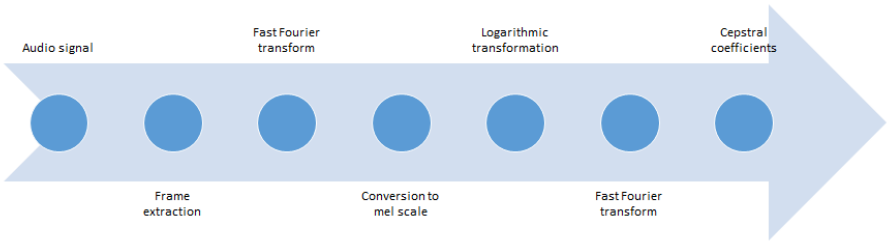


Figure 1: Algorithm for MFCC computation

The feature has a peak in correspondence of periodic elements in the original time signal.

Spectral flatness or tonality coefficient $\in [0, 1]$ quantifies how much a sound is similar to a white noise. A high value suggests that the spectrum has a equal amount of power in all spectral bands as in white noises. On the contrary, a low spectral flatness indicates that the spectral power is concentrated in a relatively small number of bands. It can be defined as:

$$flat(i) = \frac{\sqrt[L]{\prod_{n=0}^{L-1} x_i(n)}}{\frac{\sum_{n=0}^{L-1} x_i(n)}{L}}. \tag{4}$$

The **spectral centroid** indicates where the centre of mass of the spectrum is located [2]. Its mathematical expression read

$$centroid(i) = \frac{\sum_{n=0}^{N-1} b(n) f_i(n)}{\sum_{n=0}^{N-1} b(n)}, \tag{5}$$

where $b(n)$ is the amplitude of bins the audio frequencies are divided into and $f_i(n)$ is the frequncy at the center of the bin for frame i.
Its deep relationship with the brightness of a sound makes this feature an appropriate characteristic to examine.

Spectral roll-off measures the right-skewness of the power spectrum [2]. It is the 85th percentile of the power spectrum or, equivalently, the frequency below which the 85% of the total spectral energy lies.

A final remark: the feature extraction allows us to generate a large amount of information. Consequently, as first attempt to reduce the dimensionality, we produce the mean and the variance of the features of frames within the same song.

2 Methodology

2.1 Feature extraction

Feature extraction often demands a vast amount of cpu power since it executes many operations to extract a suitable representation of the data. It is advisable to use existing libraries since they have been widely tested and optimized. Consequently, we decided to rely on the functions provided by librosa library. It is the most used audio library since it is fast, well documented and works with several audio formats.

Feature extraction shrinks the dimension of the dataset by a factor of one thousand: we were able to represent the original 1.2 Gb dataset with a 1Mb csv file.

We designed a function to extract the features given the path of a song that computes the features of each frame and aggregates them to describe the whole song adequately.

Algorithm 1 provides a schematic overview of the feature extraction process implemented.

Algorithm 1 Feature extraction

```
function FEATURE EXTRACTION(paths, features)
    for path in paths do
        audio_file ← read(path)
        samples ← sample(audio_file)
        initialize feature_matrix
        for feature in features do
            add feature(samples) to feature_matrix
        end for
    end for
    feature_matrix ← aggregate(feature_matrix)
    return: feature_matrix
end function
```

2.2 Feature selection

There are two main reasons to run feature selection: it aims to filter out useless or redundant information and prevents the curse of dimensionality.

Most feature selection techniques have been developed for supervised learning since they evaluate the predictive capability of a feature. When faced with an unsupervised learning task, such as clustering, the range of features selection techniques is narrow. A common feature selection strategy relies on correlations. It aims to discard correlated features since they convey similar information but increase the noise.

Several coefficients measure the correlation but most of them have a severe drawback. They work correctly if the data are normally distributed which is hardly the case. Consequently, we used Spearman's correlation coefficient which is distribution independent. It is simply the Pearson's correlation coefficient of two random variables X and Y computed on the ranks:

$$\rho(X, Y) = \frac{cov(rk_X, rk_Y)}{\sigma(rk_X)\sigma(rk_Y)}, \tag{6}$$

where rk denotes the ranks and σ the standard deviation.

2.3 Dimensionality reduction

The principal component analysis (PCA) is an unsupervised machine learning technique to reduce dimensionality. It detects the directions along which the variance is maximized and projects the data onto them. The projection is mathematically expressed by a linear transformation which, in turn, is represented by a rotation matrix $W \in \mathcal{R}^{n \times d}$, where $n < d$ is the reduced number of dimensions.

PCA solves an optimization problem to find the best possible rotation matrix W and the inverse transformation $U \in \mathcal{R}^{d \times n}$. It computes the optimal transformations to map the data into a lower-dimensional space and to reconstruct it.

In mathematical terms, the optimization problem reads:

$$\underset{W \in \mathcal{R}^{n \times d}, U \in \mathcal{R}^{d \times n}}{\operatorname{argmin}} \sum_{i=1}^{m} ||x_i - UWx_i||^2, \tag{7}$$

where $||\ ||$ is the L_2 norm and m is the number of datapoints [3].
It has been shown in [3] that the optimal solution satisfies two conditions:

1. $W = U^T$
2. U is an orthonormal matrix

Hence, the optimization problem becomes

$$\underset{U^T \in \mathcal{R}^{n \times d}, U \in \mathcal{R}^{d \times n}}{\operatorname{argmin}} \sum_{i=1}^{m} ||x_i - UU^T x_i||^2. \tag{8}$$

After some simplifications that involve the computation of UU^T, it can be expressed as :

$$\underset{U \in \mathcal{R}^{d \times n} : U^T U = I}{\operatorname{argmax}} \ trace(U^T \sum_{i=1}^{m} x_i x_i^T U). \tag{9}$$

Finally, it has been proved that the solution of this optimization problem is the matrix U whose columns are the first n eigenvectors of the matrix $A = \sum_{i=1}^{m} x_i x_i^T$ corresponding to its n highest eigenvalues [3].

The principal component analysis is considerably expensive in computational terms: its time complexity is $\mathcal{O}(d^3 + md^2)$. The leading term $\mathcal{O}(d^3)$ derives from the computation of the eigenvectors, while the optimization runs in $\mathcal{O}(md^2)$.

For big data, this computational complexity is prohibitive. Consequently, we decided to reduce the number of features d in the preprocessing step before applying the PCA.

2.4 Clustering

The term clustering describes a family of unsupervised machine learning techniques that aim to divide data into groups called clusters. To obtain meaningful partitions, they minimize the intracluster distance and maximize the interclass one.

Even if plenty of distance measures exists, spark ml library supports only the euclidean and the cosine distance, thus reducing the viable candidates. To

choose the most appropriate distance measure it is also advisable to take in due consideration the type of space in which features are spread in. The feature space representing songs is euclidean since every point is a vector of real numbers. Therefore, we opted for the euclidean distance which ensures clear interpretability and allows computing meaningful centroids.

Defining the centroids is not a straightforward task since it depends on the type of feature space. If the feature space is euclidean, as in our case, then the average of all points in a cluster is the centroid. Hence, it may not be one of the original points. On the contrary, if non-euclidean spaces are involved, the centroid must be a point of the dataset.

Centroids represent the prototypical elements of clusters. Moreover, they shrink the representation by partitioning the space into Voronoi cells. The centre of a cell coincides with the centroid and points falling into a cell belong to the same cluster.

There exist two families of clustering algorithms, which follow to two different strategies: hierarchical or agglomerative algorithms and point assignment algorithms. The former produce a series of results in which the number of clusters varies, the latter produce just one division into a fixed number of clusters. Moreover, hierarchical algorithms may take two distinct approaches: bottom-up and top-down. Top-down clustering starts from a single cluster and splits it, while bottom-up starts with singleton clusters and merges them until only one remain.

We decided to follow the top-down approach since it is computationally more efficient than the bottom-up. Bottom-up clustering has a time complexity of $\mathcal{O}(m^2 \log m)$, while the top-down bisecting k-means has a complexity of $\mathcal{O}((k-1)m)$. Since the number of clusters is significantly lower than the data (i.e. $k << m$), the bisecting k-means outperforms the bottom-up clustering by far.

The space complexity of bisecting k-means is linear in the number of points and clusters, therefore it is $\mathcal{O}(m+k)$. Notice that the pca remarkably diminish the need for space since they reduce d, the multiplicative factor of $m+k$, from 55 to 3.

Algorithm 2 Bisecting k-means

```
1: function BISECTING K-MEANS(k, maxIter)
2:     Cluster all points in a single cluster
3:     while !convergence do
4:         for i=0; i < maxIter; i++ do
5:             Select the cluster with highest euclidean distance
6:             Find its two subclusters using 2-means
7:             Split the cluster in the new subscluster
8:             Evaluate the sum of squared euclidean distances
9:         end for
10:        Select the split minimizing the sum of squared distances
11:    end while
12: end function
```

Algorithm 2 uses the k-means algorithm with $k = 2$ to split the clusters.

K-means algorithm is a popular non-hierarchical clustering technique propose in 1957 by Stuart Lloyd. It uses iterative structure that leads to good solutions in a reasonable amount of time.

Formally, suppose to seek k partitions $P_1, ..., P_k$ of the space, k-means minimize the objective function f:

$$f(P_1, ..., P_K) = \min_{\mu_1,..,\mu_k \in \mathcal{R}^d} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mu_i)^2. \tag{10}$$

It returns the optimal centroids $\mu_1, ..., \mu_k$ that are the mean of the points of the clusters:

$$\mu_i = \frac{1}{C_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}. \tag{11}$$

Clustering is very complex problem: it NP-hard and also its approximation within some constant is NP-hard. To lower the compuatational complexity, the procedure described in algorithm 3 faces a tradeoff: it may not find a better solution at each iteration but makes the problem solvable in polynomial time. Still, it is not possible to know the number of iterations needed for the algorithm to converge to the optimal solution.

Spark relies on Lloyd's implementation of k-means since it is computationally efficient. It has a time complexity of $\mathcal{O}(mk)$ and space complexity is equal to that of bisecting k-means.

Algorithm 3 K-means

```
1: function K-MEANS(k, maxIter)
2:     Initialize k centroids
3:     Cluster all points in a single cluster
4:     for iter=0; iter < maxIter; iter++ do
5:         for i=0; i < m; i++ do
6:             Assign i to the cluster with closest centroid
7:             for j=0; j < k-1; j++ do
8:                 Centroid j becomes the average of points assigned to its cluster
9:             end for
10:        end for
11:    end for
12: end function
```

The initialization step is the most crucial operation to produce meaningful clusters. Two main strategies have been developed to solve the issue: random initialization and k-means++. The former selects uniformly at random the centroids, the latter is far more complex. It chooses a point at random and then iteratively detects the optimal initial centroids. We provide its schematic summary in algorithm 4.

Algorithm 4 K-means++

```
1: function K-MEANS++(k)
2:     Select a centroid uniformly at random
3:     for i=0; iter < k-1; i++ do
4:         Select the farthest point from the closest centroid already chosen
5:     end for
6:     Return the chosen centroids
7: end function
```

Evaluating the clustering results is a difficult task since it is not possible to compare the them with ground truths. To overcome the problem, in literature coherence measures have been proposed. Among them, the silhouette coefficient has been studied since it measures how much points are similar to their own cluster in comparison with other clusters [4]. The silhouette measure takes values in an interval [-1,1]. Higher values suggest that clusters are well separated and meaningful within their task.

The silhouette score is defined by a intracluster distance coefficient, as follow:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i,\, i \neq j} d(i, j) \tag{12}$$

an intercluster distance metric:

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j). \tag{13}$$

Then a scaling phase is necessary

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{14}$$

such that $s(i)$ belongs to the interval $[-1, 1]$ [4].

We average the score to produce a single value that merges all the components

$$s = \frac{1}{m} \sum_{i=1}^{m} s(i). \tag{15}$$

3 Experiments

The experimental procedure is divided into three phases: features extraction and selection, dimensionality reduction and clustering, as figure 2 shows.

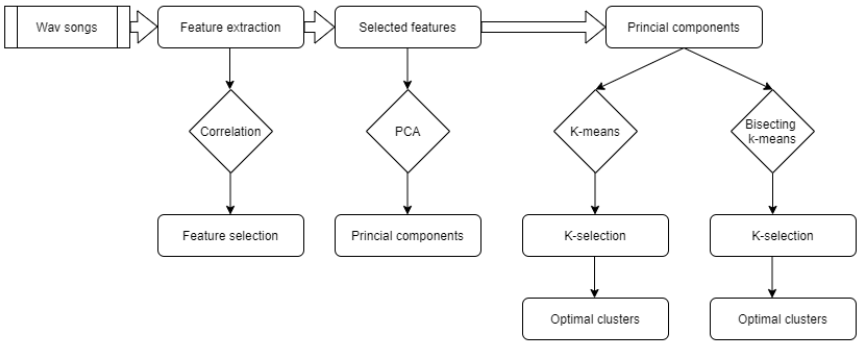


Figure 2: Schema of the analysis

Choosing a priori the best feature selection technique has proven to be a difficult task. Initially, we resolved to select features with the highest variance. Although it is a standard procedure, it has some drawbacks in this context.

Features have different scales and therefore was not possible to directly compare their variances. Moreover, features with high variance are also deeply correlated. Those characteristic mislead the pca into selecting just one component that was not able to capture the presence of clusters.

Consequently, we decided to look for uncorrelated features that could lead to more than one component and spread the data better on the cartesian plane. The feature selection procedure filter out features whose correlation with all others is above 0.5. After the preprocessing only six features remain: spectral_flatness_mean, chroma_sftf_var, root_mean_square_var, mfcc2_mean, mfcc4_mean, mfcc20_mean.

We report the heatmap of the correlation coefficients among the selected variables in figure 3.

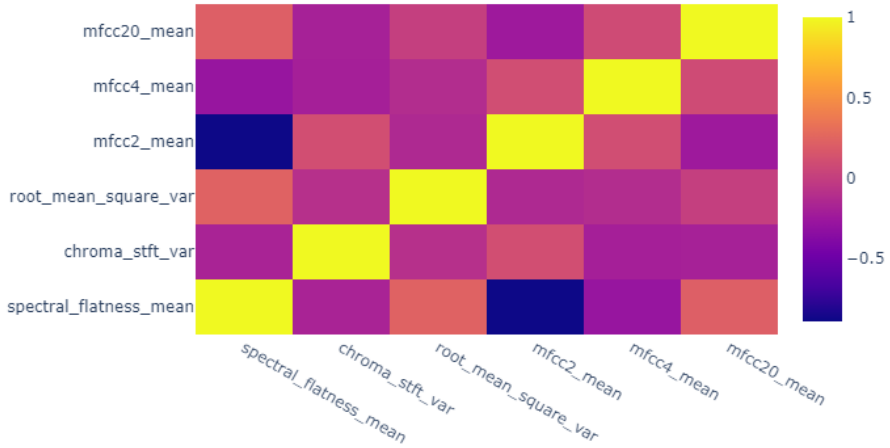


Figure 3: Heatmap of Spearman's correlation coefficients of selected variables

The feature selection reduces remarkably the number of features and prevents the arousal of the curse of dimensionality. Still, six dimensions do not allow to visualize the clusters.

We decided to reduce further the dimensionality by aggregating the features. As previously stated, the principal component analysis fits the task perfectly and can be easily applied.

The analysis of the cumulative variance clearly shows that two components are enough to explain 99% of the variance (figure 4). Moreover, they allow us to produce uncomplicated but meaningful scatterplots in which clusters clearly emerge.

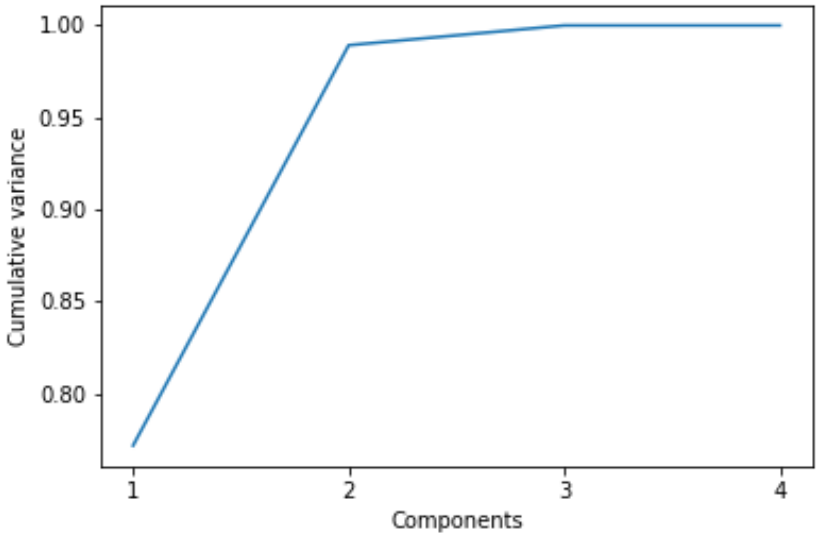
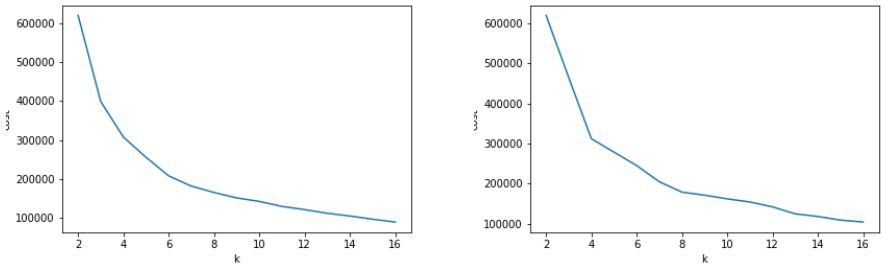


Figure 4: Cumulative explained variance of pca components

Determining the right values of k is the major challenge in clustering. It imposes considerable computational costs and cannot be completely automated.

By an a posteriori evaluation, we can determine which k is the most reasonable to represent the optimal partition. For each k, a spark function computes the costs of a division into clusters. Once the cost curve has been plotted, it is quite straightforward to appreciate which k produces the optimal partition. Right before it, the cost decreases sharply. This fact hints that the new number of clusters captures the underlying number of groups in the data. After it, the reduction in costs is less significant. New clusters do not induce a better partition but just physiologically decrease the total cost by adding centroids.

A suitable measure of cost is the sum of square distances between points in a cluster and its centroid. Spark provides a built-in function to compute it and therefore we used this metric.



(a) Costs of k-means

(b) Costs of bisecting k-means

Figure 5: Plot of costs

Bearing in mind the dissertation about the selection of k, from figure 5 clearly emerges that the optimal number of clusters for the k-means algorithms is six, while it is eight for the bisecting k-means.

4 Results

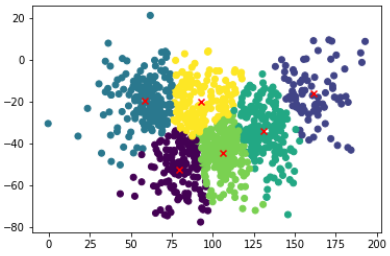
Having found the optimal values of k, we run the k-means and bisecting k-means to produce the results. We report the results in figure 7 as well as a graphical representation of the divisions in genres.

Most of the genres overlap and are positioned in the centre of the plot. Still, three genres seem to emerge from the disposition of points. On the upper left, there are sparse points, that represent classical songs. On the upper right side of the plot, reggae songs lay. Metal songs are positioned near the bottom.

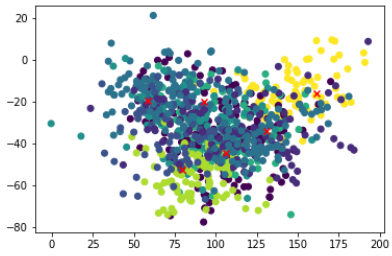
Even if clusters cannot be interpreted directly as a division into classes, we believe that a good clustering results should mirror the division in genres or in groups of genres.

As stated before, the silhouette is an appropriate metric to evaluate the clustering. The scores clearly show that k-means outperforms the bisecting k-means. Indeed, the former have a score of 0.53 while the latter reaches 0.42.

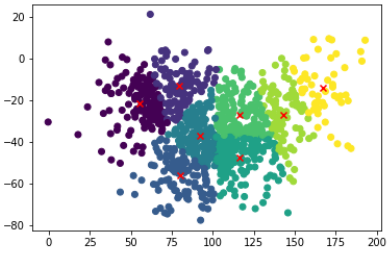
The decrease in performance of bisecting k-means may come from the division of the purple cluster on the top-right of plot (a) into two separate clusters: the yellow one and the adjacent green one in plot (c). The points in the green cluster are divided by a white portion of space and this increases the intracluster distance and reduces the intercluster distance.



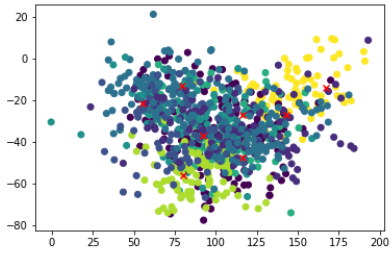
(a) Optimal clusters computed by k-means algorithm



(b) Genres in the optimal clusters computed by k-means



(c) Optimal clusters computed by bisecting k-means algorithm



(d) Genres in the optimal clusters computed by bisecting k-means

5 Conclusions

Overall, we can state that there exist three genres (classical, metal and reggae) that are different from the others. K-means finds two clusters that overlap with the classical and metal songs. On the other hand, bisecting k-means is only capable of detecting the classical song. Still, both fail to group correctly metal songs. This may explain the better silhouette scores of k-means.

The main issue of the project concerns the spatial distributions of songs: most of them lay in the centre and only a few genres are separated. To overcome it, we searched the literature about audio features [2, 5, 6]. It turned out that we have already used all the class of features available.

Consequently, it may be possible to implement and experiment with other dimensionality reduction techniques such a t-SNE. It may be able to spread more clearly the data on the plane through non-linear combination of features and let the cluster emerge more clearly.

However, we are not confident that the t-SNE may find significantly better solutions. Indeed, as an experiment, we applied the polynomial expansion to

the features before selecting them. Even the linear combination of non-linear features led to unsatisfactory results that are not reported in this review.

References

[1] *http://marsyas.info/downloads/datasets.html*

[2] Aisha Gemala Jondya, Bambang Heru Iswanto. 2017. *Indonesian's Traditional Music Clustering Based on Audio Features*

[3] Shai shalev-Shwartz, Shai Ben-Dadid. 2014. *Understanding machine learning*

[4] Peter Rousseuw. 1986. *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis*

[5] Olivier Lartillot, Petri Toiviainen. 2007. *A Matlab toolbox fir musical feature extraction from audio*

[6] Daniel McEnnis, Cory McKay, Ichiro Fujinaga, Philippe Depalle. 2005. *JAUDIO: a feature extraction library*