Computational Intelligence for Optimization

2023/2024

Vitor Souto .. 20230548
Maria Cruz .. 20230760

GitHub repository: https://github.com/vitorsouton/cifo

Statement of Contribution: Each member contributed equally for the execution of this project

Contents

1. Objective

The objective of this project is to develop a music recommendation system that leverages Genetic Algorithms (GAs) to cluster music tracks based on their attributes and recommend similar tracks to users.

2. Conceptualization

Fitness

The fitness function is designed to minimize the sum of squared distances between data points (each representing a music track) and their respective centroids (labels), using Euclidean distance. It is defined within the 'Individual' class.

The fitness value is the sum of squared distances (inertia) between each data point and its nearest cluster. Lower fitness values represent better clustering, making this a minimization problem.

The goal is to minimize the sum of squared distances by making the clusters as tight as possible and, for that purpose, we add the Calinski-Harabasz score as a penalty multiplier to the inertia.

The aforementioned score measures the ratio of the within-cluster variance to the between-cluster variance.

Representation

Given that our goal is to group similar music tracks together so that recommendations can be made based on these clusters, the representation is defined by the number of centroids. Each centroid is represented by a list of coordinates, with each coordinate being a vector of 7 digits (one for each variable from the dataset).

The optimal number of centroids was found a priori, using the "elbow method" after several KMeans runs.

Evolution process

The evolution process happens on main.py. The script has the feature of logging several relevant aspects of the evolution process and saving them to a .pkl file, for posterior analysis.

We leveraged the power of multiprocessing to handle parallel execution of multiple populations at the same time.

There are two distinct execution possibilities on main.py: the "preliminary" mode, which was used as a means for exploratory analysis, because it runs the algorithm with all 18 possible combinations of the operators, and the "run" mode, which runs the algorithm with the specific combinations of the operators that were identified as the best.

3. Methodology

The data was sourced from a Kaggle dataset that contains Spotify tracks along with their audio features. Simple data cleaning, most notably outlier removal, was performed to ensure its usability.

The features we considered for characterizing each track are: duration (ms), loudness, tempo, valence, acousticness, danceability and energy.
The functions and classes used were heavily inspired from the charles package, developed during the semester, with some adaptations to our problem.
For reference, the best run of the KMeans algorithm had an inertia of 24506.52.

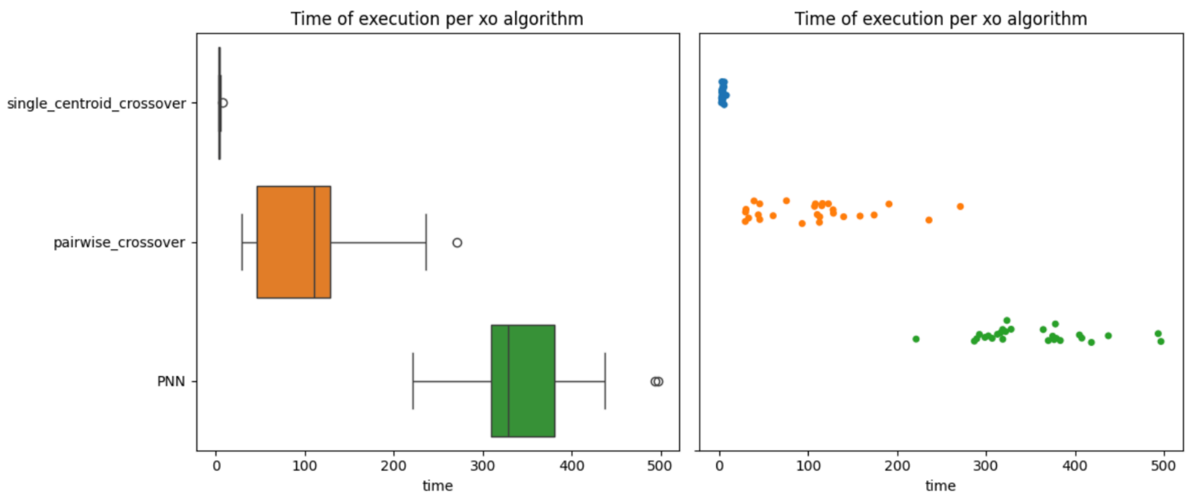
4. GA operators

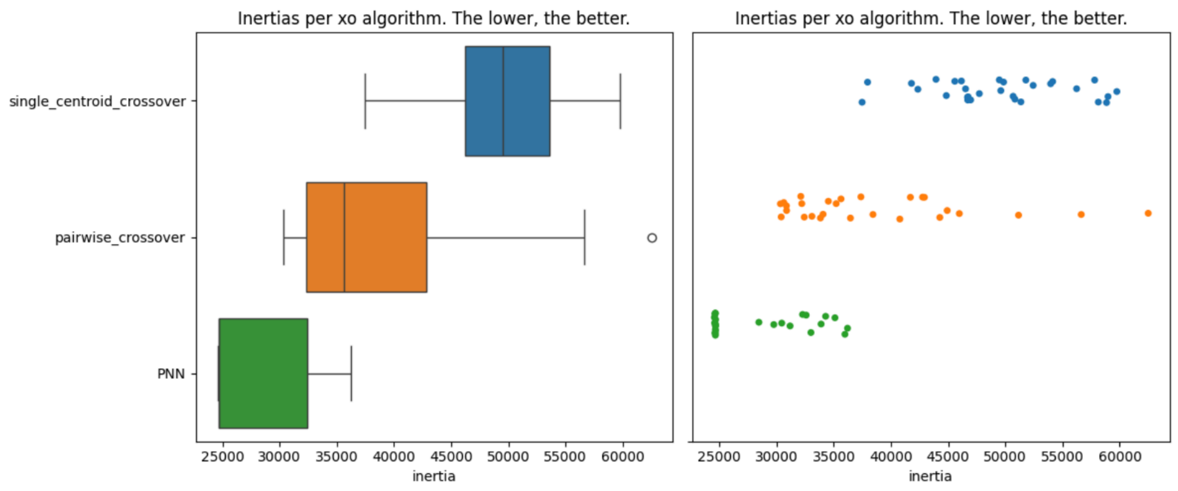
4.1. Selection

Three main selection algorithms were tested: tournament selection, roulette wheel selection and elitist selection.
The elitist selection is a variant of the tournament selection but, instead of selecting the best individual out of a random sample, a random individual is selected from a sample with the n best individuals in the population.
There was no significant fitness or time of execution differences based on the selection algorithm.

4.2. Crossover

Three main crossover algorithms were tested: single point, pairwise and pairwise nearest neighbors (PNN).
Pairwise crossover consists in bundling both parents together (therefore doubling the centroids in the representation), recalculating the distance to each point and reassigning the clusters and merging the closest centroids together. This process is repeated iteratively until the original representation shape is restored.
For PNN, the same steps are performed, with the addition of a centroid fine adjustment before merging. This adjustment is done by using 1 iteration of KMeans.
The selection of the crossover step proved to be the most impactful on the final results; both in the fitness and in the time of execution, as shown below:





4.3. Mutation

Two methods were tested: single coordinate and random swap mutation.
The former consisted in randomly selecting a gene (corresponding to a centroid coordinate) and randomly changing it to any number between 0 and 1. The latter consisted in randomly selecting two coordinates of the same gene and swapping their position.
There was no significant impact in the mutation choice, other than less variance of fitness with the random swap one.

4.4. Elitism

Elitism was always used in the runs.

4.5. Early stopping

A simple stopping condition was coded to avoid running evolution steps after 10 consecutive generations with no improvement.

5. Best solutions

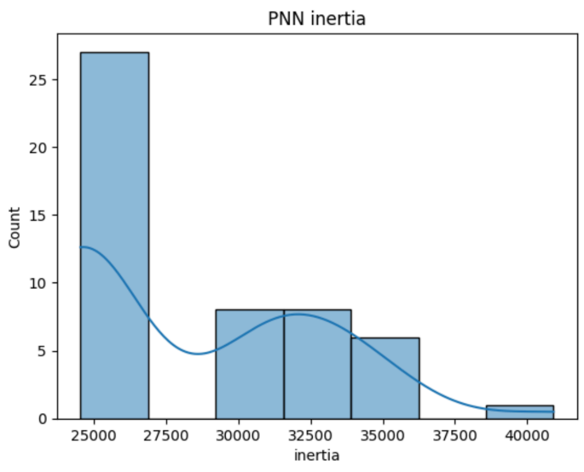
All runs searching for the best results were performed using elitist selection, PNN and random swap mutation.

5.1. Using inertia as fitness

The first idea was to use only inertia as fitness, by doing that, we would have a direct mean of comparison between the GA and KMeans.

Results

The best solution found had an inertia of 24533.45 (24506.52 KMeans best reference) took around 120 seconds to run.
In a total of 50 runs, the average inertia was 28317.25 with a standard deviation of 4387.65 and the algorithm converged, on average, in 7 generations.

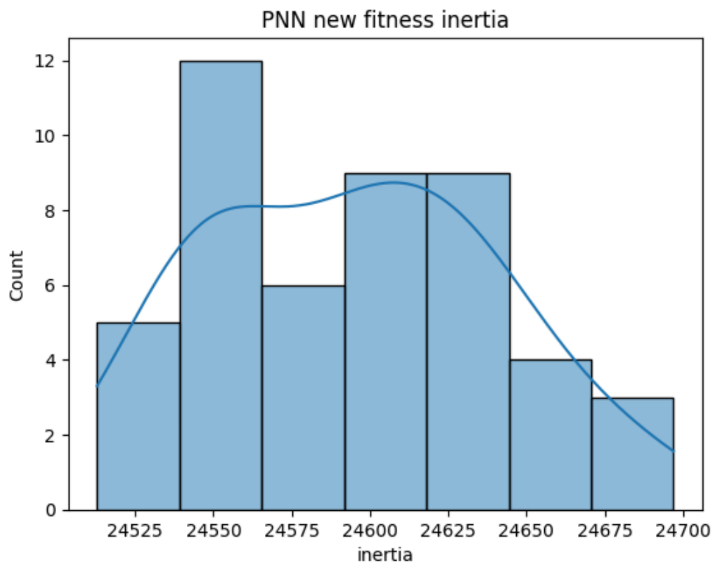


5.2. Using inertia with a penalty as fitness

As mentioned before, we decided to incorporate a penalty for solutions that yield clusters with high variance, in other words, clusters which were too dispersed.

Results

The best solution found had an inertia of 24512.87 took around 120 seconds to run.
In a total of 50 runs, the average inertia was 24594.52 with a standard deviation of 46.51 and the algorithm converged, on average, in 7 generations.

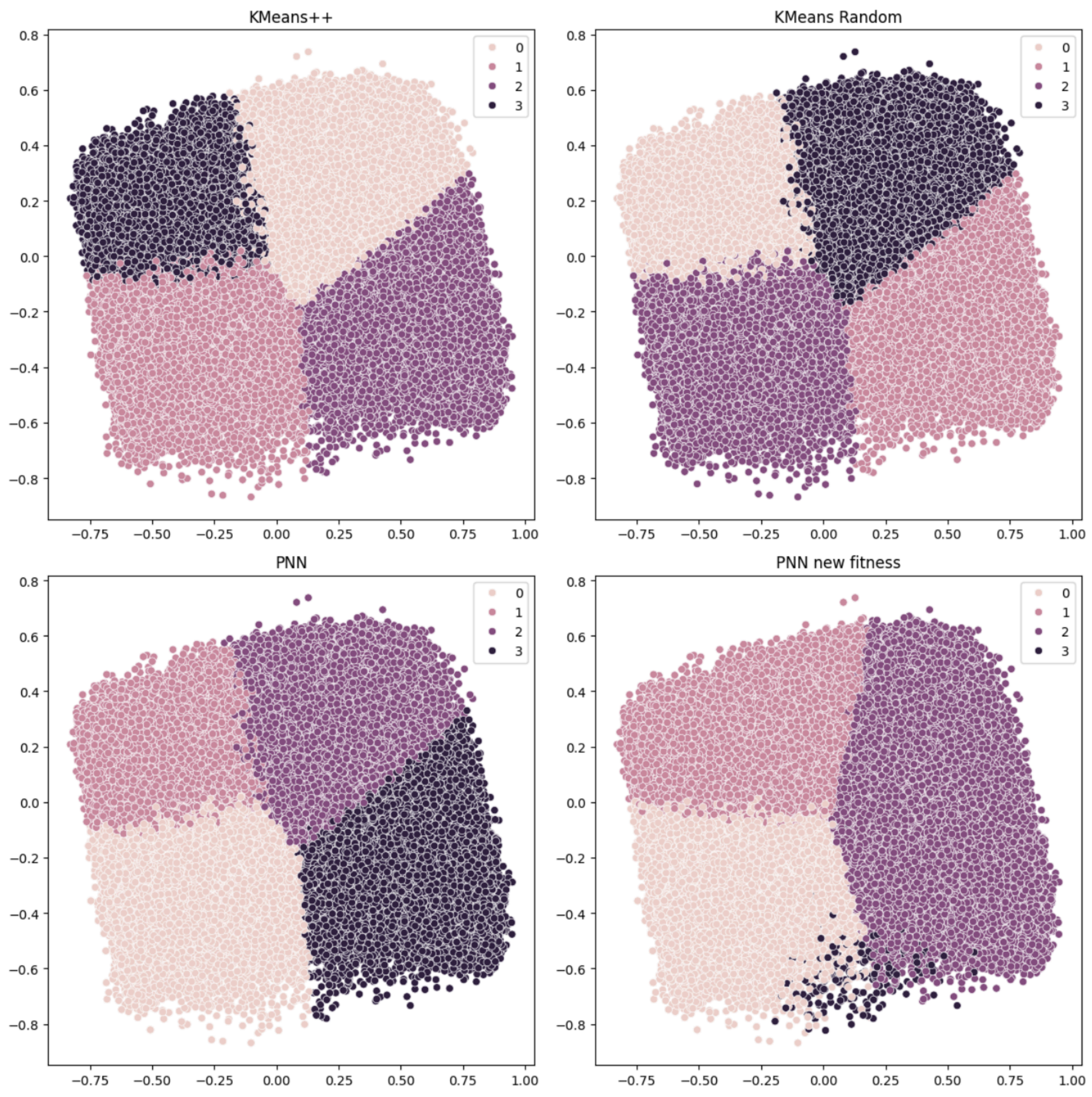


The most interesting information, however, is that this result returned a solution in which 2 centroids were identical. This means that the algorithm found a better solution with 3 clusters, instead of 4.

Therefore, we ran a KMeans initialized with 3 clusters and its inertia was 27807. In that way, we (inadvertently, but surely) found a better 3 cluster solution than KMeans.

6. Cluster analysis and visualization

Finally, we performed a Principal Component Analysis in order to reduce the dimensionality of our data and be able to plot it:



As one can see, the PNN with inertia as fitness found a really similar solution to the KMeans and, perhaps most interestingly, the PNN with the penalty fitness merged 2 clusters.

7. Conclusion

We were able to find a good way of clustering our music by means of GAs, with metrics comparable to KMeans, despite being a lot slower.
The GA proved versatile by providing a solution with a lower number of clusters, something that the KMeans algorithm is not capable of doing at all.
In order to further improve our investigation, the next step would be to experiment with different parameters for crossover and mutation probabilities, as well as the number of individuals in the selection algorithms.
Further improvements surely can be done in the parallelization or optimization of execution as a whole.