Music Classification and Nearest Neighbors Music Transition

Arthur Bonetti
Master in Finance - Financial Entrepreneurship and Data Science
University of Lausanne
Lausanne, Switzerland
arthur.bonetti@unil.ch

***Abstract*—Free Music Archive (FMA) is a dataset containing thousands of music sample and metadata. This paper uses different classification methods such as Logistic Regression, K-Nearest Neighbors, Tree Classification, Majority classification, Multilayer Perceptrons and Support Vector Classification to classify musics into their correct genre. We analyse the dataset to find the best predictive variables. Then, we implement a Nearest Neighbor algorithm to find musics that give a smooth transition between two pre-selected tracks.**

***Index Terms*—Free Music Archive, Music Classification, Logistic Regression, KNN, Tree Classification, Multilayer perceptrons, Support Vector**

I. Introduction

Music is an interesting topic for machine learning and classification algorithms. Indeed, music is basically data. We can describe a music by the length of the sound waves or its specific tone, the bit rate, and a lot more of features. Each feature will change according to the music and the genre of the music. Musics from the same genre will have similarities in the features, making it possible to use classification algorithms and expect high accuracy. On top of that, music can also be described by its metadata. A music has a published date, an artist, an album, a duration, a language and contains a lot more information. With such amounts of data available, scientists from around the world collected music samples and gathered them in some gigantic datasets. Some examples of these datasets are GTZAN, TagATune, Million Song, or even the Free Music Archive. With these datasets, it became easier for scholars to analyse music and make some insightful discoveries.

The topic of science and music is so hot that even challenges exist. This is the case of the Million Song Dataset Challenge, where the concurrent where asked to predict which songs an user will listen to. This type of challenge is clearly the basis of technology that we use in our everyday life. We can think of Spotify that implemented such features and recommends musics, artists or album, based on personal preferences and previous played songs. This challenge was created 9 years ago and gathered 150 teams of researchers. The contestants were given the full listening history for 1M users and half of the listening history for 110K users. The aim was to predict the missing half. The topic of science and music is nothing new but still interests today. Some projects like Magenta push the boundaries of the feasible further away. With today's technology, it is possible to make music using Machine Learning. Existing music datasets sense the need for music data and keep to update themselves. We can think of the Free Music Archive Next Project which aims at improving and relaunching the Free Music Archive.

II. Research Question

A. Problem

This research paper looks upon two majors problematic. The first one is the question of music classification. There are a lot of genres and sub genres in music. From major genre such as Hip Hop, Electronic or Pop to much smaller genre such as Down Tempo and Chip Music, it is not always easy to classify one music into a specific genre. The second problem this paper looks upon is the transition between two musics. Indeed, people often have a music in mind that they want to listen. However, depending on the situation, it is not always a good time to switch directly from the music of Titanic to Daddy Yankee. Similar literature on machine learning on the Free Music Archive dataset can be found from the main paper of the dataset [1]. There are other papers that intended to classify the genre of a music using various method, this is the case for example of a paper that uses Transfer Learning for music classification and regression tasks [2].

B. Objective

The objective of this paper is, first, to use Classification algorithms to find the most efficient classifier to predict a music genre. To do so, we will analyse the data to find the best predictive variables and compare the performance of some of the best know classification algorithm. We will compare the performance using two sets of variables. The second objective of this paper is to implement a Nearest Neighbour algorithm to find the songs that make a good transition between two chosen songs. To this end, we will compute the euclidean distance between the two points and do a Nearest Neighbor algorithm in the chosen direction.

C. Scope

This project is created on the scope of the Free Music Archive dataset. The computations were done on the small

subset of the database. The small subset already contains 8,000 musics with the related features and metadata. The scope of this project can easily be extended to the medium and large subset of the database. We used the binder repository available from the GitHub page [3] of the Free Music Archive to ease the installment procedure and the database manipulation.

III. Methodology

A. Database

Various databases are available for music analysis. After analysing each one of these, we find that the best documented and complete one for the purpose of genre classification is currently the Free Music Archive. With the data being already prepossessed and a jupyter notebook available to guide the steps of the researcher, the Free Music Archive is the perfect candidate in the scope of this project. The git hub repository is well maintained and contains all the information necessary to guide the user with the manipulation of the data. Furthermore, the database is already sliced in three categories; small, medium and large. In our analysis, we limited ourselves at the small subset of the database as we consider that 8,000 observations can already give meaningful results and to gain on computation time.

The database also contains sounds sample that can be used to listen to the music analysed which gives a welcome down to earth feeling. We recommend to use the binder repository as data can weight as much as one therabyte for sound samples and it is possible to experience corruption of the database while extracting it. Using the binder repository is the best way to use rapidly this database without experiencing the trouble of setting up the database, cleaning it and downloading a lot of data. However, this solution has its limitations. The memory on the session is limited to 2GB and it is very difficult to import its own environment to use TensorFlow for example.

B. Data visualisation

The database contains a lot of data and information. Therefore, before tackling the classification algorithms, we visualize the data to get an idea of the division of the number of music in each genre. We also visualizes the different features available to use as predictive variables. This graphical interpretation is done with pyplot, a library of data visualisation in Python.

C. Classification Algorithms

The principal aim of this paper is to classify the songs to their correct genre. Because there are more than sixty genres, and some with very few observations, we decide to classify the songs in the main sixteen parent genre. We use the scikit library to implement the different classification algorithms. The methodology for each algorithm is quite similar. First, we define the X variables and the y variable. We use X variables that are a subset of the features of each music and the y variable is the parent genre of the music. We will also use X as the full features to see whether more variables improve significantly the accuracy of the model. Once the X and y are chosen, we split the data into training and test data. We use a 80/20% ratio and a random state of 0 to have consistency among the result of the different algorithms.

Then we create the model, fit the data and predict the values for X_test. After that, we compare the predicted value to the true value and compute the confusion matrix, accuracy score, precision score, recall score and f1 score. For some models, such as the KNeighbors Classifier, we add a step that normalizes the data. For classifiers that require the use of parameters, we test different parameters to find the one with the best score output. We implement the following algorithms using the scikit library:

- Logistic Regression
- KNearest Neighbors Classifier
- Decision Tree Classifier
- Voting Classifier with LR, KNN and DTC
- Multilayer Perceptrons Classifier
- Support Vector Classifier

D. Music Transition

The last part of this project is about using K Nearest Neighbors to determine a smooth transition between two musics. This is done by first determining a step, or a number of songs to have as a transition. Then, we divide the distance between the two points by the step size. We go from the first point in the direction of the second point and, at each step, we perform a nearest neighbors algorithm to find the closest point. We then save this point in an array and fetch the name and ID of the song represented by the point. Once we have collected all the songs, we show the plot and the list of songs that connects our initially chosen two songs. We also create a small program that allows to directly listen to samples of the song chosen by the algorithm.

IV. Implementation

This project was programmed using the environment offered on the binder repository from the git hub of the Free Music Archive. To have a full list of the dependencies necessary to access the database, please refer to the git hub repository of the Free Music Archive. The classification algorithms and the split and train process was done using scikit.

A. Feel the data

The database contains a lot of important information and it is important as a first step in this project to have. a feel of the data. The better way to understand the data we are about to manipulate is to plot some graphs. The first question we can ask ourselves is how many tracks contains the database. As mentioned earlier, the database is categorized in three subsets; small, large and medium. The small subset contains 8,000 tracks and is the one we are going to use in this project. The medium subset is composed of 25,000 tracks and the large subset is composed of 106,574 tracks. The corresponding playtime of each subset is 2.8, 8.7 and 37.0 days.

There are more than sixty genres in the database and each one is classified in a specific category belonging to a parent category. There are 16 parent categories. The three majors categories with a number of tracks between 30,000 and 40,000 are Experimental, Electronic and Rock.

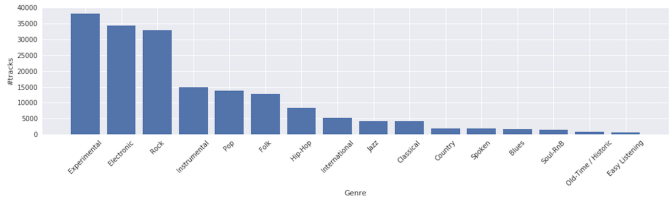

Fig. 1. #tracks by parent genre

Then, we are interested in the features of the song. Each song has 9 features. We want to plot the features to see if they are discriminant. To improve the visualisation, we keep only two genres, Instrumental and Electronic. We plot each feature in a 2D grid with each category having his own color. Instrumental is plotted in red and Electronic in blue.

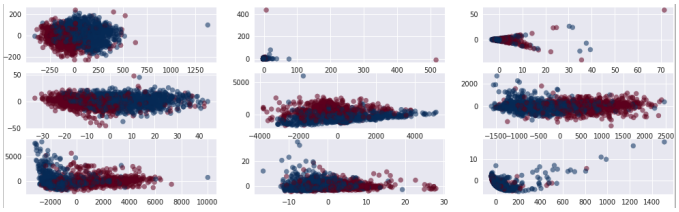

Fig. 2. Music features Instrumental and Electronic

The feature that appears to be the most discriminant is the first one on the top left which is the MFCC, which is the timbre texture feature. The 4th graph which represents the spectral contrast also seems to be discriminant. We can see a clear separation between the red and the blue color. With this preliminary analysis, we can say that the MFCC is a good choice to use as X variable to predict the genre of a music. The analysis was done using only two genres and can be pushed further. Nonetheless, now we have a hint of what we are looking for. For the implementation of the classification algorithm, we will first use the MFCC to predict the genre and then, we will also use the combination of all the features

To have an even deeper feel of the data, one can observe the jupyter notebook[4] available on the github repository of the Free Music Archive. This notebook gives even more insights on the data available and the structure of the database.

B. Quality Measures

Our project will use and compare a number of different classification models. Therefore, we need a mean to compare the results obtained for each model and to determine which one is the most efficient to classify the musics in their own genre. To this end, we use the sklearn.metrics library. For each model, we compute the confusion matrix, accuracy score, precision score, recall score and f1 score. The confusion matrix allows to see clearly the number of observations correctly classified, the number of observations misclassified and in which category it was misclassified. The accuracy measures the classifier's ability to put data points into the right class. Precision reflects the classifier's ability to correctly detect positive data points, in other word it is the True Positive / (False Positive + True Positive). Recall reflects the classifier's ability to detect all positive data points; True Positive / (False Negative + True Positive). Finally, the F1-Measure combines both the precision and recall in an harmonic mean.

C. Classification Models

To implement the classification algorithms, we use the scikit library and the packages already made. The main task when implementing the models is to choose the X and y variables, split the data into training and testing and fit the model. Some models require no parameters such as the Logistic Regression but some models require to find the good pair of parameters such as the Multilayer Perceptrons Classifier. Models such as the kNN Classifier requires the data to be normalized.

When implementing the models, we decide to choose y as the parent genre to determinate. From the small subset of the dataset, 8,000 musics are classified in 8 parent genres. For the X variables, we decide in a first place to use X as the MFCC features, and in a second place to use X as the whole set of features. This will allow us to determine whether more information available means more precision in the model or not.

First, we run a Logistic Regression. The Logistic Regression is a classification method that builds on multiple linear regression. It uses threshold based on the probability that the corresponding data point belongs to a specific class. It assumes that there is a linear relationship between the log odds of the data point belonging to a class and the numerical features. Another name for the Logistic Regression is the sigmoid function. The model uses a loss function than can be minimized using a gradient descent. In our project, we first run this model on the small subset of the dataset and with the MFCC features. We split the data into a 80% training set and a 20% testing set. Therefore, we have 6,400 observations used to train our model and 1,600 observations that we use to try to predict the genre. The following figure gives the different quality measure for this model.

```
811 classified correctly
789 classified incorrectly
confusion matrix
[[106  13   1  28  17  10  12   9]
 [ 14  48  15   8  45  15  12  22]
 [  8  11 117   4  18  17  12   9]
 [ 20   5   2 140   6  13  11   6]
 [ 10  30  13   5 103   7  10  17]
 [ 20  15  18  27   7 118   8   9]
 [ 25  15  29  23  19  16  36  42]
 [  9   8  13   6  10   4  11 143]]
accuracy  0.506875
precision  0.487509241062
recall  0.503986688378
f1  0.490372547571
```

Fig. 3. Logistic Regression MFCC

For a first model, we achieve an accuracy of more than 50%, meaning that, with a Logistic Regression, we are able to classify correctly more that half of the observations. Some categories did a better job at finding their genre than others. With the confusion matrix, we can see that the seventh category has more trouble to be classified correctly. The different quality measures give a result around 50%.

Numbers speak for themselves, however it is always interesting to see how the model draw the Logistic Regression. To illustrate the line fitted by the model, we kept only the Instrumental and Electronic genres, plotted the values in a two dimensional space and drew the Logistic Regression.

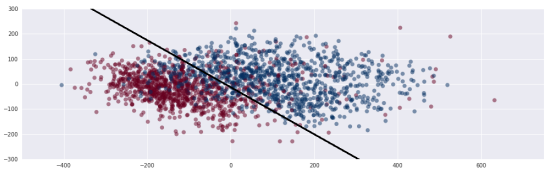

Fig. 4. Logistic Regression Illustration

We can see that, with these two categories, the line does a great job at separating the observations of each genre. The Instrumental musics are plotted in red and are more situated in the left down side of the plot and the Electronic musics are plotted in blue and are more located in the top right side on the plot. This graph is just an illustration of the Logistic Regression with two genres and not the Logistic Regression ran on the full model.

We then change the X variables to include all the features variables. The result of the model is the following:

```
926 classified correctly
674 classified incorrectly
confusion matrix
[[119  12   3  22  17   8   8   7]
 [ 15  78  17   3  24   9  15  18]
 [  3   6 127   2  24  13  13   8]
 [ 23  10   1 134   6   7  19   3]
 [  7  24  14   4 123   3   5  15]
 [ 15  12  16  24  11 128   4  12]
 [ 22  17  21  22  12  11  68  32]
 [  3  13   6   6   8   8  11 149]]
accuracy  0.57875
precision  0.572841271689
recall  0.577550004276
f1  0.571767648282
```

Fig. 5. Logistic Regression all features

This time on 1,600 observations, we classified correctly 926 observations. This is a significant improvement compared to the model with only the MFCC values. Accuracy, precision, recall and F1 are around 0.575. The Logistic Classification Model, even with its simplicity, shows good performance when classifying musics. It improved quite a bit when adding more variables to predict the genre.

The second model we are using to predict the genre of a music is the k-Nearest Neighbors Classification Algorithm. This time again, we split the data using train_test_split from sklearn.model_selection. We then normalize the X variables using a MinMaxScaler. We fit the model and predict values for y. This time however, we need to choose a number of neighbors for the algorithm to work. The k-Nearest Neighbors identifies the k closest points in the training data according to a suitable distance measure. It predicts the nominal target feature as the value that is most frequent among the k closest data points. To determine the number of k points to observe, we run a loop and see which number of k-Nearest Neighbors minimize the error. After this step, we discover that the best number of neighbors to choose is 7 for the model with the MFCC features as X variables. When running the program, we find the following results using the classification algorithm with a number of neighbors of 7 as a parameter:

```
confusion matrix
[[ 97   6   6  39   7  11  22   8]
 [ 26  41  17  12  44  11  15  13]
 [  4   5 128   8  21  14  14   2]
 [ 30   7   7 120   9  13  13   4]
 [ 13   9  18   8 118  14   6   9]
 [ 15   3  13  31   4 147   9   0]
 [ 20   8  24  44  13  17  66  13]
 [ 12   6   9  11  11  10  39 106]]
accuracy  0.514375
precision  0.516017267328
recall  0.509623987234
f1  0.503325879783
```

Fig. 6. KNN MFCC

This model outperformed a bit the Logistic Regression with

the same MFCC feature as X variables. It is interesting to notice that the distribution of observations correctly classified is different from the previous model. For example, it did a far better job at classifying the seventh column compared to the Logistic Regression Model.

Then, we repeat the process with all the features this time. Again, we loop over different number of k neighbors to determine which parameter is the most suited to the model. We find a number of neighbors equal to 9 to be the most suited for the job, which is close enough to the number used in the last model. The results obtained with this model are the following:

```
confusion matrix
[[ 74   9   9  55   9  15  16   9]
 [  9  51  25  17  16  32  18  11]
 [  4   6 123   2  14  25  16   6]
 [ 18   7   5 138   1  16  14   4]
 [  3  21  49   5  72  18  18   9]
 [ 13   3   8  19   3 165   9   2]
 [ 10  15  32  25   6  46  51  20]
 [  7   8  14   8   2  15  37 113]]
accuracy  0.491875
precision  0.494386159226
recall  0.485624658313
f1  0.476828234396
```

Fig. 7. KNN all features

This time, the model did a poorer job when adding more features. It went from an accuracy of 0.514375 to an accuracy of 0.491875. This shows us that adding variables does not always help to create a better model and therefore, the preliminary step of looking at the data and choosing a discriminant feature was useful. Not only the model is better with less variables, but it is also less costly in computation time. In any case, the model showed a worst performance than the Logistic Regression with all the features.

The next model used in this project to classify the genre of a music is the Decision Tree Classifier. This time, we have to determine the criterion to choose as well as the max_depth of the tree. A tree with a depth to deep may overfit the data and loose its prediction ability. A Decision Tree provides a sequence of binary decisions which have to me made to decide which class a data point belongs to. For our model, we decide to use the Entropy criterion, which is used for the information gain. Entropy quantifies the amount of uncertainty of a random variable Y. Concerning the depth of the tree, after some tests and trials, we find that a tree with a max depth of 6 performed a bit better than trees with other depth. We found the following results for the Decision Tree Classifier after creating the model and fitting the value:

```
confusion matrix
[[ 83   1  13  36  26   6  15  16]
 [ 20  30  22   9  46  14  25  13]
 [ 10   6  88   7  32  29  12  12]
 [ 58   3  12  79   9   4  15  23]
 [ 13  17  20   7  98  13  11  16]
 [ 30   4  28  26  19  69  24  22]
 [ 29   3  35  28  22  13  22  53]
 [ 13   7  19  11  11   8   6 129]]
accuracy  0.37375
precision  0.36820149094
recall  0.372781779282
f1  0.356993060369
```

Fig. 8. Decision Tree MFCC

This time the model did quite poorly at classifying the musics. With an accuracy of only 0.37375, the decision tree model does not seem to be very adapted to predict the genre of a music. We are still interested to know if adding more features will help to improve the Decision Tree Model. Therefore, we run the program again with this time all features taken into account.

```
confusion matrix
[[ 55   5  12  71  17  14  16   6]
 [ 13  22  31  31  34  13  13  22]
 [  3   4 118  10  34  16   5   6]
 [ 31   7   5 102  10  13  25  10]
 [ 11   6  29   7 109   7  10  16]
 [  9   3  39  43  21  87  12   8]
 [ 11   2  43  37  29  21  26  36]
 [ 11   7  12  23  13   9  13 116]]
accuracy  0.396875
precision  0.391679166004
recall  0.394293014185
f1  0.374540658842
```

Fig. 9. Decision Tree all features

The model shows some sign of improvement but still fails to obtain results similar to the two previous models. It did a good job at classifying some categories but did a very poor job at classifying others. The poor performance of the Decision Tree Model is probably due to the high number of categories we have. Indeed with 8 categories, the model fails to create a tree that is able to classify with high accuracy the musics into their correct genre.

We are now interested in seeing whether combining the three previous models allow to better results. To this end, we use Ensemble learning with a Majority Voting model. This ensemble method trains the three previous classifiers on the same training data and uses majority voting to determine the class of an unseen point. We implement this ensemble method from the sklearn.ensemble library. We use the same models as before with the same parameters and come up with the following results:

```
confusion matrix
[[125  15   5  26  12   2   4   7]
 [ 23  69  24   6  30   5  10  12]
 [  8  19 128   6  15  10   6   4]
 [ 48   8   7 119   3   5   8   5]
 [ 20  36  17   7 100   5   2   8]
 [ 31  23  14  25   8 117   1   3]
 [ 46  17  36  23  10  11  35  27]
 [ 18  18  14   6  11   7   5 125]]
accuracy  0.51125
precision  0.524435774305
recall  0.510727803578
f1  0.500921792331
```

Fig. 10. Majority Voting MFCC

The results show that even with the Decision Tree model performing a bit more poorly than the others, the majority voting did a better job overall at classifying the data. The precision of 0.5244 is the best precision score so far with only the MFCC feature as variables. We can notice that this model combining the three previous model gives more importance to the precision that the accuracy which was not always the case in the previous results. The Majority Voting model with all the features gives interesting results too.

```
confusion matrix
[[137   8  10  20   7   6   4   4]
 [ 43  73  19   3  21   4   8   8]
 [ 13  19 125   2  20   8   8   1]
 [ 51  16   8 112   2   5   7   2]
 [ 23  28  22   9 102   3   3   5]
 [ 33  14  26  20  11 111   3   4]
 [ 55  31  38  24   8  14  23  12]
 [ 21  18  13  10  11  11   8 112]]
accuracy  0.496875
precision  0.514712801551
recall  0.497571464216
f1  0.485842868819
```

Fig. 11. Majority Voting all features

Even with the Logistic Regression and the Decision Tree models being improved by taking all the features as an input, the quality measures are overall lower than the previous model. By looking at the first column of the confusion matrix, we can see that musics are too often put in the first category. The Majority Voting does not seem to be a very good fit to classify musics into their genre. So far, the Logistic Regression with all the features showed the best quality measures.

We are still interested in two more models for this first part of our project. The two models that we are going to use next are the Multilayer Perceptrons Classifier and the Support Vector Classifier. The Mulilayer Perceptrons model is built upon an Artificial Neural Network with some inputs, an activation function, some hidden layers and an output. When building a Perceptrons model, we have to choose an activation function, a solver and a number of iterations. To create the model, we use the sklearn.neural_network library. Again, we split the data into 80% training set and 20% testing set. We find that choosing an activation function has a lot of impact on the result of the model. After looking at the documentation in sklearn and trying a few activation functions. We discover that the best activation function with our dataset is the logistic activation function. The logistic sigmoid function returns f(x) = 1 / (1 + exp(-x)). Then on the topic of the solver, after testing the different parameters, we find that the 'lbfgs' solver for weight optimization returns the best result. The 'lbfgs' is an optimizer in the family of quasi-Newton methods. Finally, we use a maximum number of iterations equals to 2,000. The results of this model appear in the table below.

```
confusion matrix
[[ 98  16   1  26  14   7  28   6]
 [  8  81   9   6  29   7  25  14]
 [  4  19 109   3  18  16  24   3]
 [ 25   8   0 127   6  12  19   6]
 [  8  53  13   3  88   5  14  11]
 [ 13  22  15  27   8 113  18   6]
 [ 16  21  16  24  10  10  81  27]
 [  4  19   6   8   6   3  27 131]]
accuracy  0.5175
precision  0.52973018998
recall  0.516477756658
f1  0.519849189217
```

Fig. 12. Multilayer Perceptrons MFCC

So far, this is the best model with only the MFCC feature. The model even did a good job at predicting the seventh column that causes more problems to the previous models. When running the same model with all the features, we can observe a very interesting behaviour.

```
confusion matrix
[[ 46   0   7  92  13   2  30   6]
 [ 15   1  23  35  50  11  36   8]
 [  4   1  52  17  71   4  45   2]
 [ 42   0   5 113  13   7  23   0]
 [  6   1  23  17 112   6  29   1]
 [ 12   2  22  68  23  44  41  10]
 [ 17   0  16  80  22   7  58   5]
 [ 19   1  16  74  14  37  24  19]]
accuracy  0.278125
precision  0.287213192495
recall  0.276357261901
f1  0.245183083085
```

Fig. 13. Multilayer Perceptrons all features

This time, the model completely fails to deliver a good performance. With too many input variables, the model is confused and shows the worst performance so far. Such results indicate that when using a Multilayer Perceptrons model, in our case, it is crucial to determine the variables that show hints of good performance before running the model.

Last but not least, we use a Support Vector Classification model. We have good hope on this model as it is described as being an impressive model when classifying reasonably sized datasets. The idea behind Support Vectors is that we measure

the distance that we have to travel away from the line before we hit a data point. Then we specify a margin that should be as large as possible. This model does not require any parameter is is straightforward. Like the previous models, we split the data, create the model and fit the values.

```
confusion matrix
[[120  22   1  17  13   4  14   5]
 [ 13  81   9   4  34   7  18  13]
 [  3  10 123   5  19  13  17   6]
 [ 30   6   1 134   6   8  15   3]
 [ 11  29  12   5 108   7  10  13]
 [ 15  14  13  26   7 133   8   6]
 [ 29  20  22  15  11   6  77  25]
 [ 12  11   8   7  10   3  14 139]]
accuracy  0.571875
precision  0.571955134573
recall  0.570291995635
f1  0.569131375903
```

Fig. 14. SVC MFCC

The results are indeed impressive. With this model, we have an accuracy of over 0.57 when so far with only the MFCC features we found an accuracy of maximum 0.5175. This model outperformed all the previous models. The precision, recall and accuracy scores are all superior to previous models. We are now interested at the results when using all the features as variables.

```
confusion matrix
[[121  17   3  19  15   2  14   5]
 [  5 101  10   6  22   4  20  11]
 [  3  11 133   1  19   9  15   5]
 [ 33  10   1 137   3   7  10   2]
 [  9  30   9   2 114   2  17  12]
 [ 18  12  12  18   6 141   8   7]
 [ 22  18  21  18  10   4  87  25]
 [ 10  11   7   4   8   1  22 141]]
accuracy  0.609375
precision  0.614010198735
recall  0.608794782367
f1  0.60905354379
```

Fig. 15. SVC all features

The accuracy is over 0.60, thus crushing every other models we have created so far. The Support Vector Classification Model may be simple and does not require parameters, but it is still a very strong classification tool. The very good results of this classification method may be due to the size of our dataset which is reasonably sized.

D. Music Transition

The second part of this project is to create an algorithm that will give as an output a playlist that creates a smooth transition between two songs. To this end, we decide to implement an algorithm similar to a K Nearest Neighbor algorithm. The idea is to plot the two songs in a two dimensional grid, split the distance between the two songs in smaller interval, go one step from the first point to the second point, and perform a Nearest Neighbor algorithm to fetch the song closest to the point we ended after this first step. We repeat the process until we reach the second point.

First, we decompose the MFCC features to have two components. This step is useful to plot the data in a 2-dimensional space and to be able to easily get and create new data points. Once we have decomposed our dataset, we choose the two songs that we are interested in and save their coordinates. Then, we input the number of steps that we want, in other word it's the number of music that we want as a transition. After that, we measure the distance that we have to increase each time for the x and y values to get to the second point with the wanted number of steps. We position ourselves in the first point, which is the coordinate of the first song, and increase the x and y values a step toward the second song. We compute the coordinates where we landed and perform a Nearest Neighbors algorithm. Our Nearest Neighbors algorithm take as an input the coordinate of the point that we aim at finding neighbors, the dataset with every points and the number of neighbors. Because we are only interested in finding the closest song to the coordinate, we choose the closest point. It would also have been possible to look for a few neighbors and choose one randomly, or choose one with the most similar genre. The Nearest Neighbors algorithm requires us to to measure the distance from the other points and we used the euclidean distance for this purpose. We sort the data and return the index of the nearest neighbor.

This process is the basis of our algorithm. Then, we save the index of the nearest neighbor in an array and repeat the process with the following step. At the end, we find ourselves with an array full with indexes values. We use these indexes to search for the name of the song and show it to the user. We also plot the process in a graph to get an idea of the whole process.

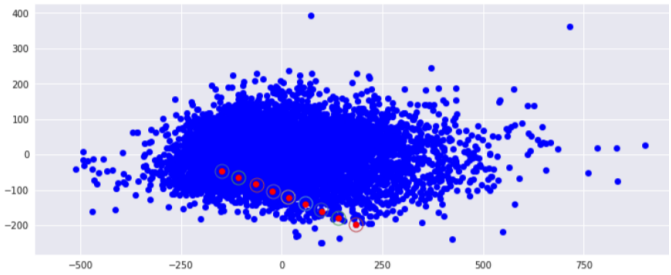

Fig. 16. Music Transition Illustration

The blue points represent each music in the small dataset of the Free Music Archive. The red points represent the new point coordinate found after each step. The circles around each point are the position of the closest observation to our coordinate. In this example, we can typically see that the two songs where far apart in this MFCC scale, with our algorithm, we were

able to find songs that joined them with a smoother transition. The final result is a list of Song names and indices.

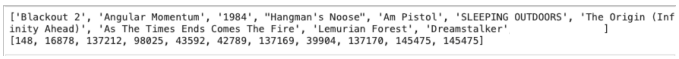

Fig. 17. Music Transition Results

We also were also able to write a few lines of code that allow the user to listen directly to samples of the songs. However, in order for this program to run, the user need to download the music samples from the Free Music Archive. The folder containing these samples weights more than 7GB. The jupyter notebooks usually limit the user to less memory usage than that.

V. Results

This project had 2 objectives : Predict the genre of a music and create a transition between two musics. For the genre prediction, we compared various classification methods on our dataset. We found that the best predicting model was the Support Vector Classifier with all the features variables as an input of the model. We found an accuracy of 0.609375, a precision of 0.614, a recall of 0.60888 and a f1 of 0.609 for our best model. Those results are pretty good when considering the particularity of the data observed. With a dataset of only 8,000 observations, with 6,400 observations used for training, the results are beyond expectations. One way to improve the model may be to try a Gaussian Classification method or a Deep Neural Net. It also may be interesting to try prediction on the whole dataset that contains more than 100,000 observations. However, to undergo such models, the jupyter notebook set up by the Free Music Archive is not a viable option. We tried to create a Gaussian Classifier and a Deep Neural Network, the jupyter notebook limited us to 2GB of memory, not enough to run a Gaussian Process and the notebook did not allowed us to change the environment to use Tensorflow. Nonetheless, the results are pretty positive with some basics models. Most of our classification method performed around a 0.5 accuracy, which is way better than just randomly assign the observations to a category.

The second objective of this project was to create a transition between two musics. Again we can say that, even with a simple model, the objective was met. The Nearest Neighbor algorithm appears to be very adapted to this task. The implementation was straightforward and the output met the expectancy. However, it is hard to say if the musics chosen do a great job at creating an harmonic transition. This may depends more on the listener. It is still a great tool to give ideas of songs that are closely related in the MFCC feature. It may be used by DJ to give them ideas on how to connect two musics. The only drawback is that, in the small dataset of the Free Music Archive, most musics are not popular musics and they almost always do not appear in Spotify or related music streaming application. And because the data folder containing the music samples uses 7GB of memory it is not possible to import it in the jupyter notebook. It is kind of disappointing to solve the problem of the music transition but not be able to listen to the result. Nonetheless, the model can always be extrapolated and used with another music dataset.

Overall, this project has met the objectives it had aimed for. The results are very satisfying and, even if improvements can still be done, we were successfully able to implement the ideas we had in mind.

VI. Conclusion

We can think of music in a lot of way, and with this report, we proved that music can be represented and visualized with data. During this report, we spent more hours looking at numbers, graphs and table than listening to music. Machine Learning tools and Data Analysis are powerful tools that allow us to see the world from a different perspective. Even by knowing little about music, a data scientist can help a DJ or a music maker in his task. The models and algorithm implemented in this project are one solution possible and not the only way to solve our problem. During this project, we were guided by our intuition and mostly worked with errors and trials. We were always very curious about the results when fitting a model to the data and were never able to predict the success rate of a classification method before launching the program. This is the reason that pushed us to try the most algorithms possible to find the better one for this job.

The Free Music Archive is a great dataset with a lot of resources and examples that really help understanding the data and the structure of the dataset. When looking back at the beginning of this project, we are glad to have choose this dataset for this project. The binder link to the jupyter notebook with libraries and data already set up was really a big help to get going on the project. Even with the limitations of the jupyter notebook it allowed us to do most of the task we wanted without the trouble of downloading a lot of data and dependencies.

A few leads for improvement of our model may be to try to set up the classification models on the whole dataset rather than on the small subset with 8,000 observations. Another suggestion is to try to use the metadata and add it to the X variables. For some models, more observations meant more accuracy, by adding a song metadata such as the language, the nationality of the singer, the length of the song or the date of publication, the performance of the classifier may be improved. It would also be interesting to implement more complex algorithms using the neural network technology. For the song transition, it may be interesting to use another variable as a dimension and compare the results obtained. It is also possible to increase the number of k Nearest Neighbors to obtain multiple recommendations instead of just the closest song.

We were very glad to undertook this project and obtain these results. To conclude this project, we will look at the data and enjoy some graphs of the 'See You Again' song by Wiz Khalifa.

REFERENCES

[1] Defferrard, Michaël, et al. "Fma: A dataset for music analysis." arXiv preprint arXiv:1612.01840 (2016).

[2] Choi, Keunwoo, et al. "Transfer learning for music classification and regression tasks." arXiv preprint arXiv:1703.09179 (2017).

[3] https://github.com/mdeff/fma , last consulted the 6 June 2021

[4] https://nbviewer.jupyter.org/github/mdeff/fma/blob/outputs/analysis.ipynb , last consulted the 6 June 2021