Classification of Audio Features Using Machine Learning Algorithms

Pawel Cislo
Faculty of Engineering, Environment and Computing, Coventry University
MSc Data Science and Computational Intelligence (ECT104) Stage 1
Coventry, United Kingdom
cislop@uni.coventry.ac.uk

***Abstract*—In this paper, the primary objective is to perform classification on audio features of music files to define the core characteristics of various musical genres and test the performance of classification algorithms in the music industry. Musicians tend to try to sound similar with their style to a particular music type or artist, thus, the raw characteristics of a genre could be a useful guideline for new composers. The project uses data from Million Song Dataset (MSD) and applies classification algorithms using Python programming language with the use of scientific libraries.**

Keywords—classification; machine learning; million song dataset; music; python

I. Introduction

Getting precise data can be difficult when it comes to the features of sound (waveform) or video image. Services like Spotify or YouTube try to fill this kind of data automatically for the use of recommendation system using many algorithms. Music industry uses music information retrieval (MIR) system that can recommend songs based on attributes such as genre, rating and beats per minute with the use of implemented algorithms to extract descriptors of music [1].

Analysing the MSD dataset can provide crucial information for artists and music services. The information retrieved by using supervised learning method like classification analysis, could present the similarities between the particular genres and define the audio characteristics of the artists and their styles.

This paper is organised in the following way:

- Chapter II introduces the review of literature
- Chapter III covers the description of the dataset used in the classification process
- Chapter IV covers the process of applying machine learning algorithms
- Chapter V presents the results of experimenting on a dataset
- Chapter VI gives a brief conclusion and discussion of the project.

II. Literature Review

The publication of MSD by Thierry Bertin-Mahieux and Brian Lamere [2] was a significant advantage for the music industry. It is the most extensive available dataset in the field that can be used for many kinds of exploration and feature evaluation. MSD is a result of collaboration between Laboratory for the Recognition and Organization of Speech and Audio (LabROSA) and The Echo Nest. The authors propose various ideas for applications of the dataset, such as metadata analysis, artist recognition or automatic music tagging. More information about the project is located at the MSD website with code samples, tutorials or FAQ section. Apart from the dataset and the website, the authors present the example of practical application to predict the year of song release basing on the audio features of the collected data. For the benchmark process, Bertin-Mahieux and Lamere use *k* nearest neighbours (*k*-NN) method as a first approach and Vowpal Wabbit (VW) as the second approach. The results of the prediction are presented in Table I – "average absolute difference and the square root of the average squared difference between the predicted release year and the actual year", where it is clear that VW achieved the best score [2].

TABLE I. Results on year prediction on the test songs [2]

Method	**Diff**	**Sq. diff**
constant pred.	8.13	10.80
1-NN	9.81	13.99
50-NN	7.58	10.20
vw	**6.14**	**8.76**

Short after the release of MSD in the year 2011, group of Carnegie Mellon University students published a research article "Music Genre Classification with the Million Song Dataset 15-826 Final Report". The group of students classify music genre using "several broad feature classes to capture different acoustic, musical, and textual aspects of the data" [3] on the MSD. The authors split the dataset by artist for improved results into three groups: training, tuning and test sets. After that, Liang et al. developed a "blend model" that contains several models like Hidden Markov Model (HMM), Lyrics bag-of-words or Canonical Correlation Analysis (CCA) [3]. The group combines audio and lyric features using CCA. To present the final results, the researchers use “two different timbre HMM’s, bag-of-words lyrics, lyric sentiment, and loudness and tempo features” [3] and illustrate them on numerous graphs with one of the conclusions that pop and rock genres are characterised rather by cultural style and history than by acoustic features.

The research article “Evaluation of different audio features for musical genre classification” by Baniya et al. published two years after the publication of MSD in 2013 presents different approach to classify music genre by the evaluation of audio features. The group of students from the Republic of Korea worked on GTZAN dataset by George Tzanetakis, which consists of 1000 records divided into ten genres. For the experimentation, the students applied Extreme Learning Machine (ELM) with bagging and compared the results to Support Vector Machines (SVM) in terms of the accuracy. Music features have been classified into four general groups: dynamics, rhythm, spectral and harmony. The results in Figure 1 present that spectral features have the highest impact on the classification of a genre. The accuracy of ELM reached 85.60%, and the score did not differ much from SVM classifier, which scored 86.10% [4].

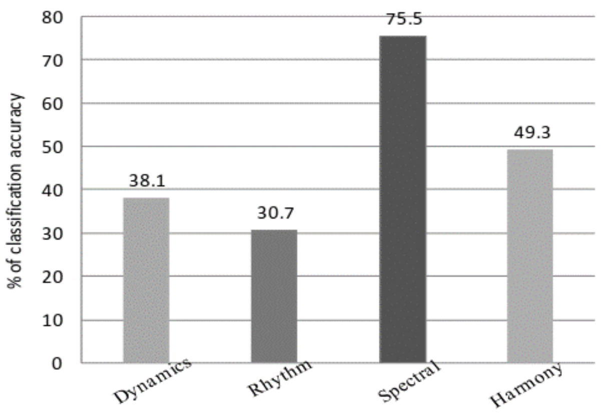


Fig. 1. Music genre classification of four different sets of Features [4]

The book “Fundamentals of Music Processing” by Meinard Müller published in 2015 covers many MIR tasks that are the core of music processing and information retrieval. Müller collects lots of information and presents them in one book, so students do not need to look into many research papers. The presented techniques apply to the processes of classification and analysis problems. Furthermore, MIR introduces the possibility to find the use of machine learning on audio features [5]. Owing to the popularisation of MIR with the publication of Müller’s book, it is possible to perform more convenient analysis using ready-made software, programming libraries or web applications. Librosa is one of the example packages for Python programming language that has been developed by McFee et al. in 2015 using the concept of MIR.

III. Dataset Description and Features Selection

The dataset used for experimentation comes from open source collection of metadata and audio features published in 2011 by Bertin-Mahieux et al. in the Proceedings of the 12th International Conference on Music Information Retrieval.

The original MSD contains the following data:

- “280 GB of data
- 1 000 000 songs/files
- 44 745 unique artists
- 7 643 unique terms (Echo Nest tags)
- 2 321 unique musicbrainz tags
- 43 943 artists with at least one term
- 2 201 916 asymmetric similarity relationships
- 515 576 dated tracks starting from 1922” [2].

Because of the size of the dataset, for faster performance, this project uses a subset of 10000 songs that is also available from MSD website. The data is initially stored in Hierarchical Data Format (HDF5), which is designed for storing large datasets. To use this kind of data, Python requires additional libraries that could complicate the project and affect the results. To solve any potential problems, the files have been converted to Comma-separated values (CSV) format by Ryan Whitcomb who adds and removes some attributes from the original dataset [5].

The database consists of 35 fields, which consist a string, real and integer values. The sample record with its attributes and values is presented in Appendix A (three additional columns were added to define the data type, value range, existence of blank values and the description of the attribute taken from MSD website).

The attributes of each record can be separated into two groups:

- metadata (data used to identify the content)
- audio analysis (data about the audio) defined by the Echo Nest Analyze API.

For the requirements of this project, the metadata is not taken into the process of classification. The machine learning algorithm is used on numerical data collected by The Echo Nest, which is presented in Table II.

TABLE II. Audio Analysis Data Selected for Classification Algorithm

artistHotttnesss	loudness
barsStart	mode
beatsStart	startOfFadeOut
duration	tatumsStart
endOfFadeIn	tempo
familiarity	timeSignature
key	

The data about the confidence of measurement provided by The Echo Nest have been omitted, as it is not relevant for this analysis. The field “songHotttnesss” has been skipped as well as it contained 4350 missing values. Data about the longitude and latitude has been dropped as well, but for future analysis, it could be used to find a cultural aspect of different countries to define the music characteristics.

MSD contains 457 unique genres in the “terms” field collected by The Echo Nest and 276 unique genres collected in the “artist_mbtags” field by the MusicBrainz community. The tags from “terms” field compared to “artist_mbtags” field seem much more relevant as they contain only five missing values, whereas MusicBrainz tags have 6289 of them.

The 20 most popular genres of “terms” field have been presented in Appendix B with the use of pandas package function, which counts the number of occurrences of unique values and sorts them in ascending order. The result show, how diversified is the dataset in terms of the class name and values.

In order to find out the most affecting features in the dataset, the model needs to be tested with feature selection algorithm. Applying feature selection and using only the most important attributes results in several advantages:

- Easier interpretation of the model
- Reduction of the variance
- Reduction of overfitting the model
- Better computational cost and time of training the model

This example uses Random Forest, which ranks how much the purity of each node is improved. Random Forest method uses Gini impurity that is a decrease in impurity of all trees. Using this model, the tree can be cut at a particular level as the nodes with the highest decrease in impurity occur at the bottom of the tree. The code presented in Appendix B ranks the importance score of every feature that sums up to 100%. As observed, the most affecting attributes are: artistHotttnesss (15%), familiarity (14%), loudness (13%) and startOfFadeOut (9%). The rest of the features scores below 9%.

As advised by MSD publishers this project uses a set of selected genres to simplify the variety of names [2]. The five chosen genres presented in Table III are taken from the “terms” field. The genres below have a similar number of records in the database that will result in a better environment for classification methods. It is essential to have a similar number of examples in each class, as it will result in a more accurate outcome. The dataset ends up with 727 records in total.

TABLE III. GENRES USED FOR EXPERIMENTS

Genre	Number of records
country rock	156
latin jazz	150
post-grunge	146
dance pop	141
gangster rap	134

IV. EXPERIMENTAL SETUP

The dataset has been initially cleaned, yet it contains a few missing values of audio features marked in Appendix A. As the dataset contained only four missing values for “familiarity” and five missing values for “terms” field, the rows with missing values have been dropped. If there is a lot of missing value, it is recommended to deal with the missing data in the other way, as by dropping records not all the information is used. The other options available for dealing with missing values are to:

- Impute missing values by replacing the blank fields with sensible values. One of the methods is to perform zero impute where the records are filled with zero value. The other way is to replace the records with the average value of the attribute
- Use algorithms, which support missing values in the dataset.

The figure below presents how the model works for this experiment. The experiment uses supervised learning, which provides immediate feedback if the predicted music genre was labelled right or wrong. Firstly, the audio features are provided, after what the machine learning algorithm (in this case classification algorithm) is applied. Afterwards, the model can be tested by providing new audio features and determining the genre of music, whether it is pop, rock or jazz.

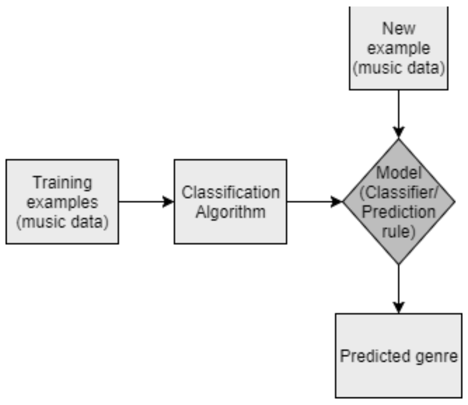

New example (music data)
Training examples (music data)
Classification Algorithm
Model (Classifier/ Prediction rule)
Predicted genre

Fig. 2. Working of the machine learning algorithm

To perform the experiments, this project uses Python programming language with the use of scientific libraries. The code is tested by using Jupyter Notebook (interactive computer environment) that supports IPython command shell for Python as well as other programming languages like R or Ruby [6].

The project runs Python 3.6 on Jupyter Notebook in version 5.2.2 installed on Windows 10 – 64bit operating system (OS). The whole package can be installed by downloading Anaconda Distribution in version 5.0.1 that is a recommended for data science industry [7]. Prior to launching the experimental environment, the OS required the installation of additional libraries for running the experiments.

In the beginning, the environment requires importing several scientific libraries that contain predefined functions to run the required operations (Appendix C). The main part of the project (classification algorithms) uses a scikit-learn library that provides required functions for Python programming language. For better understanding, the entire code on the attached images has been commented.

Secondly, Pandas package is used to load the MSD in a .csv format and assign it to a variable *df1*, which later is used for the functioning of algorithms. To confirm that the dataset is loaded properly, the function *print* and *len* are used to display the names of all the columns in a dataset and the number of rows (Appendix C).

Another step is to choose the list of attributes and the label (in this example it is the genre of music) that will be predicted in the testing period. Before that, it is highly recommended to choose specific genres for the class value, so it will be easier to find a correlation between specific types of music. Moreover, using all the genres will result in high computational cost and time (Appendix C).

After all the previous steps, the data needs to be randomly split into training and testing set. This example uses scikit-learn function *train_test_split* which shuffles the dataset and separates it into variables for attributes and labels being trained and tested. The function allows to determine the size of training (*test_size*) and testing set. In this project, 75% of data is used for training, and the rest (25%) is used for testing (Appendix C).

Successful implementation of the code allows the program to undertake classification algorithms. Predefined functions from scikit-library allow performing the classification quickly and precisely, by creating a classifier *clf* and training it (*clf.fit*) (Appendix C).

Each classification algorithm ends with accuracy test *a1*, which displays the outcome in percentage value after multiplying the result by 100.

This project implements three classification methods for the supervised learning process:

- K-Nearest Neighbours (KNN)
- Decision Tree
- Random Forests.

A. K-Nearest Neighbours (KNN)

K-Nearest Neighbours is one of the most straightforward classification algorithms to use and understand [8]. Despite this, it is a well working procedure with a satisfying prediction. KNN is recommended to be used on models where there are not too many attributes that define the class, so it seems to fit this model.

KNN works by calculating the distance between every instance and assigning the unclassified record to the nearest class (neighbourhood), for example using Euclidean distance function and classifying records to the nearest neighbours. For two points p and q, the distance d can be represented by the following formula (1).

$$\begin{aligned} d(p,q) &= d(q,p) \\ &= \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \end{aligned} \qquad (1)$$

The number of k (chosen neighbours) in the classification algorithm is significant. There is no general algorithm, which will help to determine the most optimal number of neighbours. It is a good solution to keep an odd number of neighbours as there will be no tie between choosing the class. The optimal value of the number of neighbours (k value) in the algorithm is usually set to 5 as the lower the value is, the higher is the variance, and the lower is the bias. The simple approach to select the value of k is to take the square root of the number of testing examples *n* as in the formula below (2). A small value of *k* results in a higher influence of noise on the result, on the other hand, a larger value of *k* results in increased computational cost.

$$k = \sqrt{n} \qquad (2)$$

In this example, the number of testing examples equals to a quite large number (545), what determined 23 as the value of *k*, which provided the highest accuracy.

Using KNN has several advantages that are suitable for this project. It handles well complex target function, which this model represents, as well as it uses the entire information of the dataset and the training does not require too much of the time. On the other hand, as it uses all the information, some information may be irrelevant, that is why data has been preprocessed before running this classification. Another disadvantage is that the model requires calculating the distance between each record in the dataset, what increases computational cost.

The code used to run and test the classification is presented in Appendix C.

B. Decision Tree

Another algorithm used in this project is Decision Tree [9]. It is easy to visualise classification model that arranges algorithm in a structure of a tree. The core idea is to break the dataset into small subsets based on homogeneity of examples, which are accessed by making specific decisions. The algorithm can be dealing with numerical as well as categorical data.

The final structure is a set of:

- Decision nodes – for example (music genre) that has branches (rock, pop) that lead to more specific ones (alternative rock, indie pop) by making decisions on the way
- Leaf nodes – representing decision (loudness or tempo)

Decision Tree is generated using the ID3 algorithm, which calculates information gain that informs how well records are separated into music genres using a given attribute, so it chooses the attribute with the highest information gain. The algorithm needs to calculate the amount of information (entropy) represented by the attribute. The formula (3) uses a collection *S*, where p_i represents the proportion of collection that belongs to class *i*.

$$Entropy\ (S) = \sum_{i} p_i(log_2 p_i) \qquad (3)$$

The advantage of using decision tree algorithm is its simplicity to understand and visualise, just like KNN. Another important aspect is that it does not require data preparation techniques. The module does not support blank values, which were removed in this model before the classification. On the other side, with too much of complex data, the over-complex tree may result in overfitting and does not predict the values correctly.

The code used to run and test the classification is presented in Appendix C.

C. Random Forests

The last algorithm used – Random Forests, in short, is a set of Decision Trees [10]. It is classified as one of the most accurate algorithms. As the name suggests, the algorithm creates a number of trees inside a forest. The model is more efficient with more trees, what results in a higher accuracy of the model. Random Forests classification algorithm performs on the same regulations as Decision Tree.

During the training time, Random Forests work by creating a multitude of decision trees, whereupon the output (class) is a mode of all the classes.

An example use of Random Forests is asking several friends for recommendation, whereas Decision Tree limits to asking only one friend.

The advantage of using Random Forests is its ability to deal with missing values and prevention of overfitting the model, what is not an advantage of Decision Tree. This algorithm is also used in feature selection, just like in this project.

The code used to run and test the classification is presented in Appendix C.

D. Database and Code

All the files needed to perform the experimentation have been uploaded to my GitHub profile.

The files included in the repository are:

- Jupyter Notebook (m24com_classification.ipynb)
- Dataset (musicclean.csv)

The link to my GitHub repository is:

https://github.com/pyxelr/M24COM_Classification_in_Python

V. RESULTS

As by using Random Forest Classifier, the algorithm determined the most meaningful features of audio, which are visualized on the Pareto chart below:

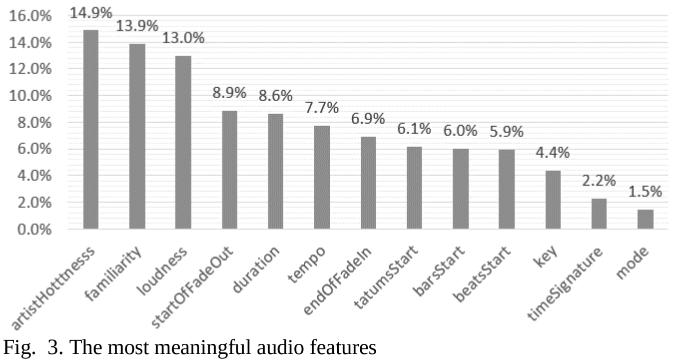


Fig. 3. The most meaningful audio features

The results state that the genre of music is mostly characterised by the:

- Popularity of the record (artistHotttnesss)
- Popularity of the artist (familiarity).

These factors determine that the artists who are on the top of the playlists have a huge impact in determining the shape of the genre with their performance. When it comes to the pure audio data, the most characteristic features are:

- Level of the sound (loudness)
- Moment when the melody starts to fade out (startOfFadeOut)
- Duration of the record (duration)
- Number of beats per minute (tempo).

The advice for aspiring artists using the collected data is to observe the performance of the most popular artists that play the desired genre. The other factors requiring paying the most attention to is playing the music at the correct sound level and tempo as well as limiting the song to a specific duration and the moment of fade-out process. Choosing the right mode (major or minor) as well as estimating the number of beats per bar is the least important characteristic.

The following table presents the results of applying different classification algorithms:

TABLE IV. ACCURACY OF THE CLASSIFICATION ALGORITHMS

Classification algorithm	**Accuracy with all 11 features**	**Accuracy with 4 most affecting features**
KNN	35%	32%
Decision Tree	50%	58%
Random Forests	55%	67%

Without a doubt, Random Forests classification outperformed other algorithms with the accuracy of 55% while using all 11 attributes and 67% with the use of only four most affecting ones. The score of 53% or 67% is still not a satisfying one what gives a conclusion that the model lack of consistency of the data or the number of examples.

By applying feature selection, the model result with reduced computational cost, as well as highly improved accuracy for all the algorithms, except for KNN where the performance decreased by 3%.

VI. DISCUSSION AND CONCLUSIONS

In general, music is one of the hardest application domains to classify. Predicting genre basing on audio features is a tricky task for machine learning as the data supplied by The Echo Nest is not that clear to understand. When it comes to attributes like "artistHotttnesss" or "barsStart" for which the average value of "barsConfidence" measures 24% in the MSD, it presents that the marking algorithm is not too confident about measuring audio data.

Getting low accuracy while the experimentation has been set up well, does not always mean that there is something wrong with the data. It is good to ask a domain expert if he could get a higher accuracy by inferring the class with the given features. If the answer is no, then getting more data will not help, but getting different features. On the other hand, if the answer is yes, then it is recommended to test the model if it scores a high variance or bias problem. High bias problem would require lowering the regularisation parameter of a classifier or changing a classifier to the one with a higher variance. In case of a high variance problem, getting more data is a solution.

Data consistency is something that this dataset lack of and this shows how much it can affect the outcome. One of the possibilities to solve this problem and get better results is to use the original instance of MSD with more values. The other solution is to try using different datasets or including more of them into a single experiment. One of the better sets of data are:

- GTZAN dataset by George Tzanetakis
- Free Music Archive (FMA) published on GitHub website as a public dataset. The dataset is a collection of data from FMA website published by a group of four students on the International Society of Music Information Retrieval (ISMIR) conference in 2017 [11].

REFERENCES

[1] Müller, M. (2015) Fundamentals Of Music Processing. Cham: Springer

[2] Bertin-Mahieux, T., Ellis, D., Whitman, B. and Lamere, P. (2011) "The Million Song Dataset - Proceedings Of The 12Th International Conference On Music Information Retrieval". 12Th International Society For Music Information Retrieval Conference (ISMIR 2011) [online] 591-596. available from <http://ismir2011.ismir.net/ISMIR2011_complete_proceedings.pdf> [29 November 2017]

[3] Liang, D., Gu, H. and O'Connor, B. (2011) Music Genre Classification With The Million Song Dataset 15-826 Final Report. [online] available from <https://www.semanticscholar.org/paper/Music-Genre-Classification-with-the-Million-Song-D-Liang-Gu/8c30fcd3e1c24debef7f1ef4aa7679fb51143eb6> [2 December 2017]

[4] Baniya, B., Ghimire, D. and Lee, J. (2013) "Evaluation Of Different Audio Features For Musical Genre Classification". Sips 2013 Proceedings [online] available from <https://ieeexplore.ieee.org/document/6674516/> [5 December 2017]

[5] Whitcomb, R. (2016) Music CSV Library [online] available from <https://think.cs.vt.edu/corgis/csv/music/music.html> [2 December 2017]

[6] Jupyter Team (2015) What Is The Jupyter Notebook? — Jupyter Notebook 5.2.2 Documentation [online] available from <https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/What%20is%20the%20Jupyter%20Notebook.html> [8 December 2017]

[7] Anaconda, Inc. (2017) What Is Anaconda? [online] available from <https://www.anaconda.com/what-is-anaconda/> [9 December 2017]

[8] Thirumuruganathan, S. (2010) A Detailed Introduction To K-Nearest Neighbor (KNN) Algorithm [online] available from <https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/> [10 December 2017]

[9] Sayad, D. (n.d.) Decision Tree - Regression [online] available from <http://www.saedsayad.com/decision_tree_reg.htm> [11 December 2017]

[10] Polamuri, S. (2017) How The Random Forest Algorithm Works In Machine Learning [online] available from <https://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/> [11 December 2017]

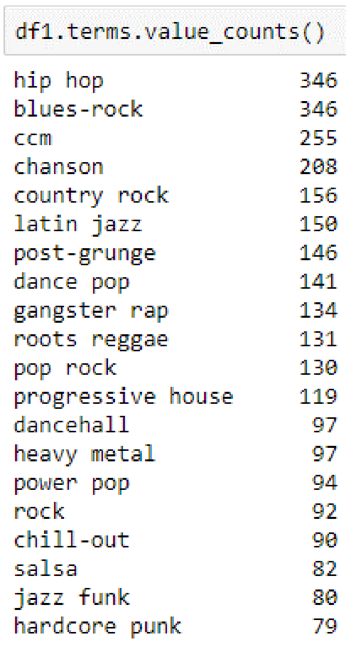
[11] Defferrard, M., Benzi, K., Vandergheynst, P. and Bresson, X. (2017) FMA [online] available from <https://github.com/mdeff/fma> [8 December 2017]

Appendix A – Dataset

TABLE V. Description of a single entry in a dataset

Key	Data type	Value	Value range (blank values)	Description from MSD
artistHotttnesss	Real	0.788805935	0 – 1.08	estimation of how popular the music record is
artistId	String	ARF2EHS1187B994F4E	-	Echo Nest ID
artistName	String	Kings Of Leon	-	artist name
artistMbtags	String	american	(6289)	tags from musicbrainz.org
artistMbtagsCount	Integer	2	0 – 9	tag counts for musicbrainz tags
barsConfidence	Real	0.109	0 – 8.86	confidence measure
barsStart	Real	1.18583	0 – 59.74	beginning of bars, usually on a beat
beatsConfidence	Real	1	0 – 1	confidence measure
beatsStart	Real	0.15226	-60 – 12.25	result of beat tracking
duration	Real	187.92444	1.04 – 1819.77	in seconds
endOfFadeIn	Real	0.328	0 – 43.12	seconds at the beginning of the song
familiarity	Real	0.845768866	0 – 1 (4)	popularity of the artist
key	Real	0	0 – 904.80	key the song is in
keyConfidence	Real	0.612	0 – 19.08	confidence measure
latitude	Real	36.16778	-41.28 – 69.65	latitude
location	String	Nashville, Tennessee	(3)	location name
longitude	Real	-86.77836	-162.43 – 174.76	longitude
loudness	Real	-5.907	-51.64 – -1.026	overall loudness in dB
mode	Integer	1	0 or 1	major or minor
modeConfidence	Real	0.735	0 – 1	confidence measure
releaseId	Integer	449471	0 - 823599	ID from 7digital.com or -1
releaseName	String	Holy Roller Novocaine	-	album name
similar	String	AR3KUCG1187B9A0BEA	-	Echo Nest artist IDs (sim. algo. unpublished)
songHotttnesss	Real	0.685940227	0 – 1 (4350)	algorithmic estimation
songId	String	SOLGHDZ12AB0183B11	-	Echo Nest song ID
startOfFadeOut	Real	180.344	-21.39 – 1813.43	time in sec
tatumsConfidence	Real	0.784	0 – 9.23	confidence measure
tatumsStart	Real	0.15226	0 – 12.25	smallest rythmic element
tempo	Real	116.823	0 – 262.83	estimated tempo in BPM
terms	String	southern rock	(5)	Echo Nest tags
termsFreq	Real	1	0 – 1	Echo Nest tags freqs
timeSignature	Real	4	0, 1, 3, 4, 5 or 7	estimate of number of beats per bar, e.g. 4
timeSignatureConfidence	Real	0.347	0 – 1	confidence measure
title	String	Wicker Chair	(1)	song title
year	Integer	2003	1926 – 2010 (5320)	song release year from MusicBrainz or 0

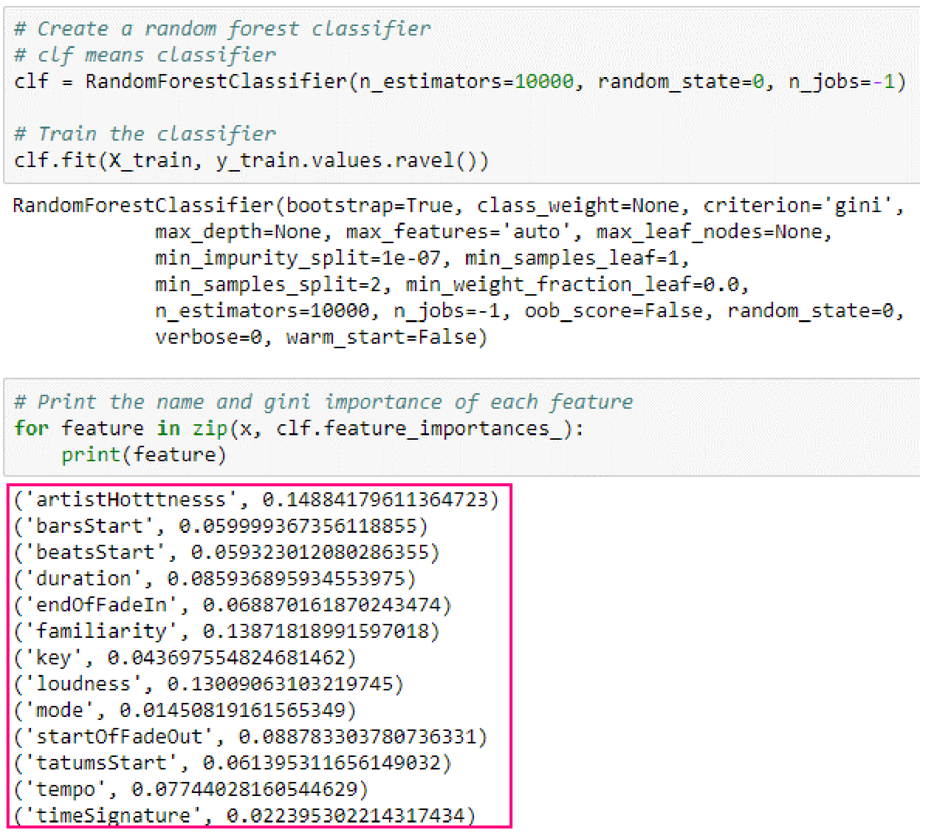
Appendix B – Feature Importance



```
df1.terms.value_counts()
```

```
hip hop              346
blues-rock           346
ccm                  255
chanson              208
country rock         156
latin jazz           150
post-grunge          146
dance pop            141
gangster rap         134
roots reggae         131
pop rock             130
progressive house    119
dancehall             97
heavy metal           97
power pop             94
rock                  92
chill-out             90
salsa                 82
jazz funk             80
hardcore punk         79
```

Fig. 4 Most popular genres in “terms” attribute



```
# Create a random forest classifier
# clf means classifier
clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)

# Train the classifier
clf.fit(X_train, y_train.values.ravel())
```

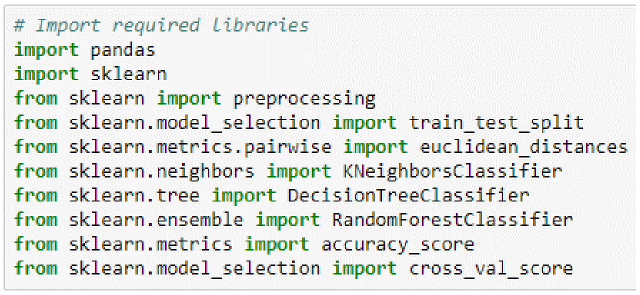
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10000, n_jobs=-1, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

```
# Print the name and gini importance of each feature
for feature in zip(x, clf.feature_importances_):
    print(feature)
```

```
('artistHotttnesss', 0.14884179611364723)
('barsStart', 0.059999367356118855)
('beatsStart', 0.059323012080286355)
('duration', 0.085936895934553975)
('endOfFadeIn', 0.068870161870243474)
('familiarity', 0.13871818991597018)
('key', 0.043697554824681462)
('loudness', 0.13009063103219745)
('mode', 0.014508191615 65349)
('startOfFadeOut', 0.088783303780736331)
('tatumsStart', 0.061395311656149032)
('tempo', 0.077440281605 44629)
('timeSignature', 0.022395302214317434)
```

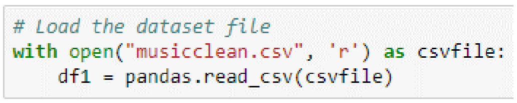
Fig. 5. Feature importance

APPENDIX C – EXPERIMENTAL SETUP



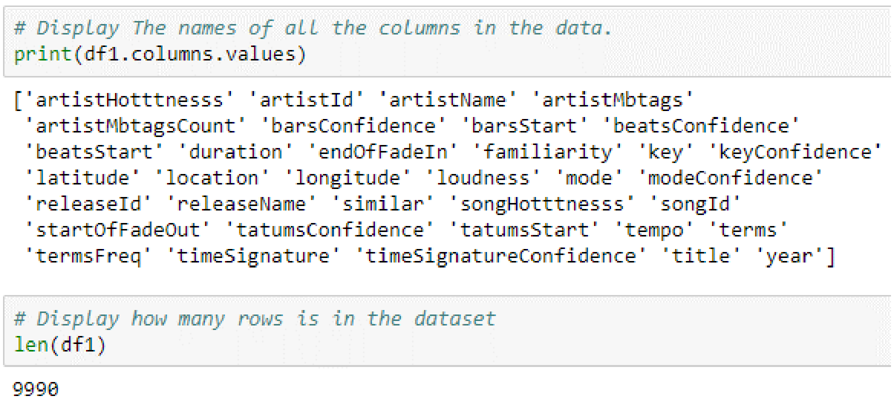
```
# Import required libraries
import pandas
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

Fig. 6. Import libraries



```
# Load the dataset file
with open("musicclean.csv", 'r') as csvfile:
    df1 = pandas.read_csv(csvfile)
```

Fig. 7. Load the dataset file



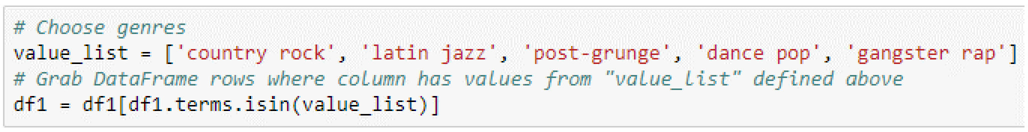
```
# Display The names of all the columns in the data.
print(df1.columns.values)
```

```
['artistHotttnesss' 'artistId' 'artistName' 'artistMbtags'
 'artistMbtagsCount' 'barsConfidence' 'barsStart' 'beatsConfidence'
 'beatsStart' 'duration' 'endOfFadeIn' 'familiarity' 'key' 'keyConfidence'
 'latitude' 'location' 'longitude' 'loudness' 'mode' 'modeConfidence'
 'releaseId' 'releaseName' 'similar' 'songHotttnesss' 'songId'
 'startOfFadeOut' 'tatumsConfidence' 'tatumsStart' 'tempo' 'terms'
 'termsFreq' 'timeSignature' 'timeSignatureConfidence' 'title' 'year']
```

```
# Display how many rows is in the dataset
len(df1)
```

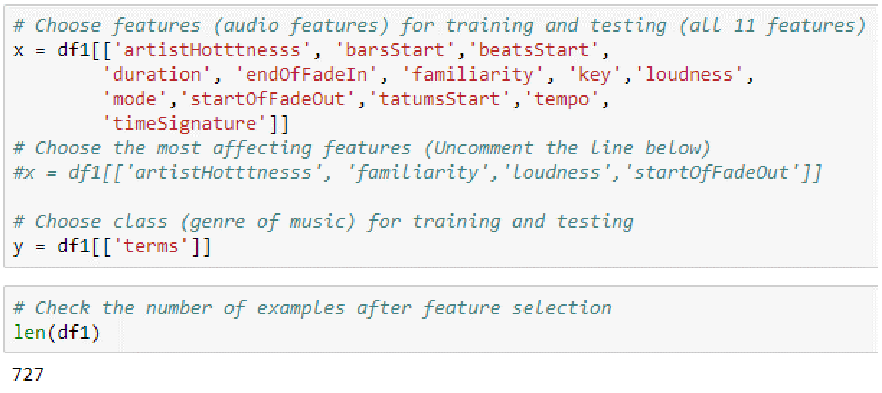
```
9990
```

Fig. 8. Display feature names and the number of rows



```
# Choose genres
value_list = ['country rock', 'latin jazz', 'post-grunge', 'dance pop', 'gangster rap']
# Grab DataFrame rows where column has values from "value_list" defined above
df1 = df1[df1.terms.isin(value_list)]
```

Fig. 9. Choose genres from a dataset



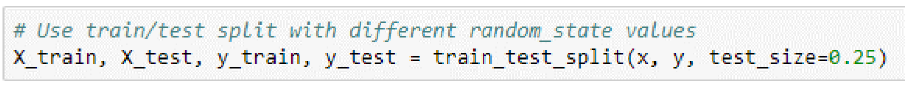
```
# Choose features (audio features) for training and testing (all 11 features)
x = df1[['artistHotttnesss', 'barsStart','beatsStart',
        'duration', 'endOfFadeIn', 'familiarity', 'key','loudness',
        'mode','startOfFadeOut','tatumsStart','tempo',
        'timeSignature']]
# Choose the most affecting features (Uncomment the line below)
#x = df1[['artistHotttnesss', 'familiarity','loudness','startOfFadeOut']]

# Choose class (genre of music) for training and testing
y = df1[['terms']]
```

```
# Check the number of examples after feature selection
len(df1)
```

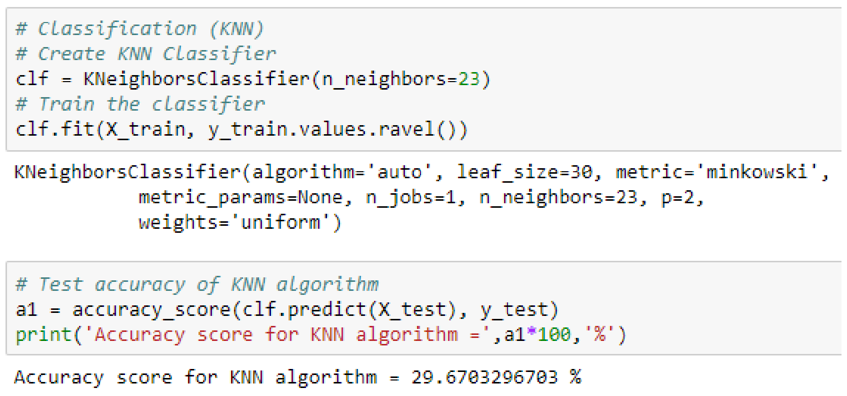
```
727
```

Fig. 10. Choose genres and display the number of rows after feature selection



```
# Use train/test split with different random_state values
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
```

Fig. 11. Split data into training and testing set



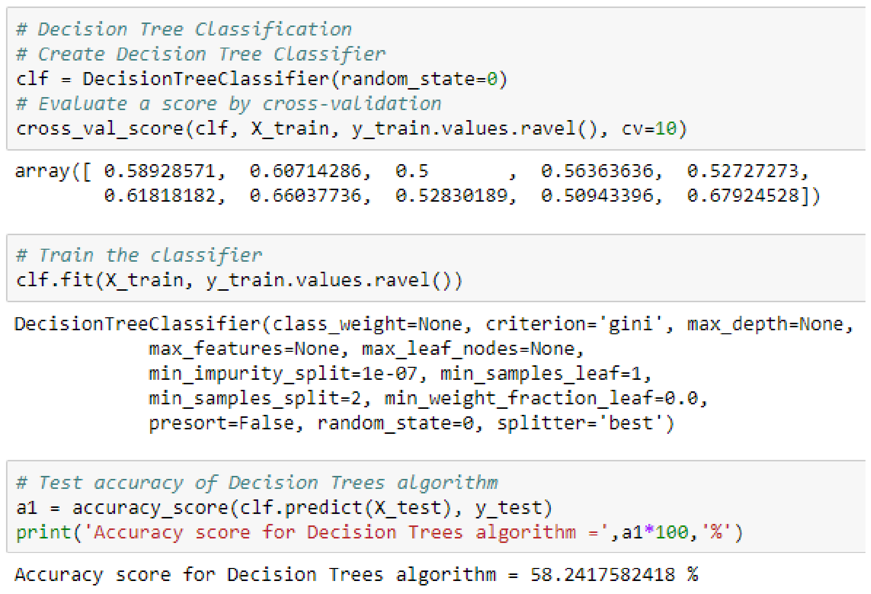
```python
# Classification (KNN)
# Create KNN Classifier
clf = KNeighborsClassifier(n_neighbors=23)
# Train the classifier
clf.fit(X_train, y_train.values.ravel())
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=23, p=2,
           weights='uniform')
```

```python
# Test accuracy of KNN algorithm
a1 = accuracy_score(clf.predict(X_test), y_test)
print('Accuracy score for KNN algorithm =',a1*100,'%')
```

```
Accuracy score for KNN algorithm = 29.6703296703 %
```

Fig. 12. Run KNN algorithm and check the accuracy



```python
# Decision Tree Classification
# Create Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=0)
# Evaluate a score by cross-validation
cross_val_score(clf, X_train, y_train.values.ravel(), cv=10)
```

```
array([ 0.58928571,  0.60714286,  0.5       ,  0.56363636,  0.52727273,
        0.61818182,  0.66037736,  0.52830189,  0.50943396,  0.67924528])
```

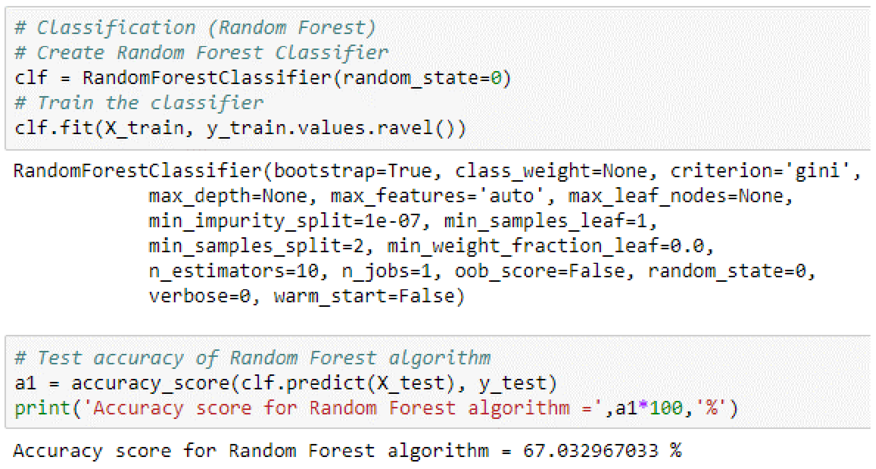
```python
# Train the classifier
clf.fit(X_train, y_train.values.ravel())
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best')
```

```python
# Test accuracy of Decision Trees algorithm
a1 = accuracy_score(clf.predict(X_test), y_test)
print('Accuracy score for Decision Trees algorithm =',a1*100,'%')
```

```
Accuracy score for Decision Trees algorithm = 58.2417582418 %
```

Fig. 13. Run Decision algorithm and check the accuracy



```python
# Classification (Random Forest)
# Create Random Forest Classifier
clf = RandomForestClassifier(random_state=0)
# Train the classifier
clf.fit(X_train, y_train.values.ravel())
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

```python
# Test accuracy of Random Forest algorithm
a1 = accuracy_score(clf.predict(X_test), y_test)
print('Accuracy score for Random Forest algorithm =',a1*100,'%')
```

```
Accuracy score for Random Forest algorithm = 67.032967033 %
```

Fig. 14. Run Random Forests algorithm and check the accuracy