Machine Learning Algorithms for Music Genre Classification

Konstantinos Kalais
University of Thessaly, Volos
kkalais@inf.uth.gr

https://github.com/Kkalais/Music_Genre_Classification

Abstract

Music genre classification is a broadly considered subject as it is an effective technique to structure and arrange the huge quantities of music records accessible on the Internet. Machine Learning methods have demonstrated to be able to distinguish such enormous pools of information, and eventually classify them.

A musical genre is described by the usual characteristics shared by its users. These attributes are connected most of the times with the rhythmic structure of the related instruments, as well as the harmonic content of the music that stands out from other genres' harmonies. Currently musical genre categorization is performed physically. Converting this manual procedure to an automatic musical genre classification can help or replace the human user in this process and would be a useful extension to music information retrieval frameworks.

In this paper, we describe the process of feature selection among a large set of features of a music archived dataset. Then, we explain how each supervised (the algorithms learn on a labeled dataset) machine learning method - we choose- classifies our samples, after analysing tracks metadata, and finally summarize a performance comparison from where we should get the most suitable technique for our problem. More specifically, classical machine learning algorithms such as Decision Trees, Support Vector Machines, K Nearest-Neighbors and MultiLayer Perceptron Neural Nets are employed. Using the proposed algorithms, classification accuracy of 74,21% for sixteen musical genres is achieved.

Keywords: machine learning, music genre classification, feature selection

1. Introduction

1.1. The problem

More than ever, the web has turned into a spot for sharing inventive work - for example music - among a worldwide network of music artists. While music and music accumulations originate before the web, the web empowered a lot bigger scale accumulations.

While individuals used to possess a bunch of vinyls or CDs, now they have moment access to

University of Thessaly.

the entire of distributed musical library by online sites. Such sensational increment in the size of music accumulations made the need to naturally sort out a gathering of music items, as publishers and users can't manage them physically any longer.

1.2. The motivation

Music genre classification is a significant activity with numerous real time applications. It is recommended that ten thousands tunes were discharged each month on web sites, for example, Soundcloud and Spotify. So as the amount of music being discharged everyday keeps increasing, the requirement for exact metadata required for music database organisation and management is expanded.

Having the option to quickly arrange tunes by genre in some random playlist or library is a significant task for any music streaming-purchasing administration, and the area for metadata analysic that complete and accurate music and audio classification gives is basically restricted.

Additionally, by classifying successfully music genres we can prompt to more customized music choices and cleared music creation services.

1.3. Related Works

Machine learning techniques for music genre classification have been studied by many.

In 2018, Hareesh Bahuleyan (1) utilizes hand-extracted features of an Audio dataset's samples and trained four traditional machine learning classifiers (namely Logistic Regression, Random Forests, Gradient Boosting and Support Vector Machines) with 65% accuracy.

Danny Diekroeger (2) tried to classify songs based on the lyrics using a Naive Bayes classifier, but concluded that analyzing solely lyrics is not enough.

In this study, we will build on top of the works done before and see if we can improve them by applying classical algorithms and neural networks on a feature selected dataset.

2. Dataset Construction and Preprocessing

Experiments were carried out on a dataset for music analysis called FMA (3) (Free Music Archive), an open and easily accessible dataset suitable for evaluating several tasks in Music Information Retrieval, a field concerned with browsing, searching, and organizing large music collections. FMA provides 106.574 music pieces - with pre-computed features and track level metadata - categorized in 16 musical genres, which we split them into training and testing set with 30% test size.

The dataset had some minor inconsistencies such as missing features that we removed during the dataset cleaning phase. It includes 252 features and after dropping the highly correlated ones - with at least 80% correlation, in order to make our algorithm more cost and time efficient - it consists of 24 features where the basic ones are explained below.

As for the **features dropping step**, when we examine the features of a dataset, some of them might not be useful to make the necessary prediction. Correlation refers to how close two vari-

Table 1. Number of songs per genre

GENRE	SONGS
HIP-HOP	891
POP	346
ROCK	3889
EXPERIMENTAL	17
FOLK	874
JAZZ	238
ELECTRONIC	2166
SPOKEN	0
INTERNATIONAL	129
SOUL-RNB	0
BLUES	66
COUNTRY	0
CLASSICAL	265
OLD-TIME / HISTORIC	357
INSTRUMENTAL	81
EASY LISTENING	0
TOTAL	9319

ables (features for our problem) are to having a linear relationship with each other. Features with high correlation are more linearly dependent (e.g. $y = 3 \cdot x$) and hence have almost the same effect on the dependent variable, in contrast with two features that are non-linearly dependent (e.g. $y = x^2$). So, when two features have high correlation, we can drop one of the two features.

We use the LabelEncoder class provided by Scikit Learn library, in order to transform the nominal values(numeric codes used for labelling or identification, such as strings, datetimes, etc) to numeric and enable the algorithm to perform arithmetic operations on these values. Next, we create the correlation matrix, which is computed as follows:

Correlation Matrix = $[Cor(x_j, x_k)]_{ik}$, where $\overrightarrow{x_j}$ and $\overrightarrow{x_k}$ are the vectors of two features' values and : $Cor(x_j, x_k) = \sum_{i=1}^{n}(x_{ij} - mean(x_j)) \cdot (x_{ik} - mean(x_k)) \cdot (\sum_{i=1}^{n}(x_{ij} - mean(x_j))^2)^{-1/2} \cdot (\sum_{i=1}^{n}(x_{ik} - mean(x_k))^2)^{-1/2}$

and visualize it through the correlation heat-map, as we can see on the figure below.

- Instrumentalness : how many vocals are present in a song
- Acousticness : the probability that the song has acoustic instruments
- Liveness : the probability the song was played in a live performance
- Speechiness : the presence of spoken words in a track
- Energy : the amount of activity and liveliness in the song
- Dancebility : the probability that the songs is made for dancing-entertaining purposes and not a calm version of music
- Valence : the musical positiveness conveyed by a track. Tracks with high value on the valence feature sound more positive and happy, while tracks with low value sound more depressed and negative.
- Tempo : the general evaluated rhythm of a track in beats per minute (BPM)

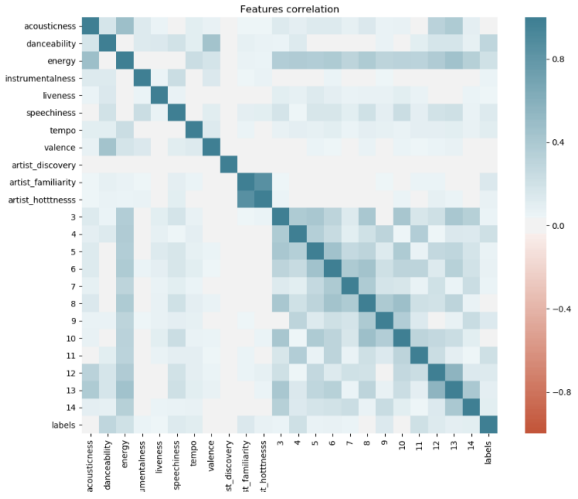


Figure 1. Plot of features correlation

Lastly, we scaled our data using the MinMaxScaler class of Scikit Learn in the range of (0,1) for easier convergence. MinMaxScaler is a good

choice if we want our data to have a normal distribution or want outliers to have reduced influence. Specifically, if max(X) and min(X) are the maximum and minimum values that appear in all dataset for a given feature X, a value X is replaced by newX using the equation:

$$newX = \frac{X - min(X)}{max(X) - min(X)}$$

This procedure was not implemented for the SVM methodology, as SVMs are scale invariant.

3. Classification Methods

3.1. General Approach and Algorithms used

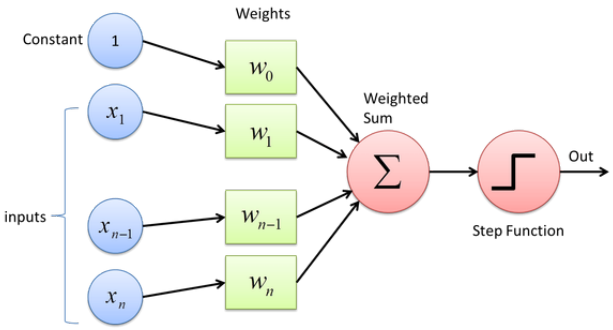
Based on the papers we mentioned, we decided on the use of eight algorithms to focus on the issue:

- Multilayer Perceptron
- Random Forest
- Support Vector Machine
- Gradient Boosting
- K-nearest neighbors
- Logistic Regression
- Linear Discriminant Analysis
- XGBoost

We will analyse below the work implemented in each one.

3.2. Multilayer Perceptron

Neural networks are built as the model of neurons present in the human brain. Neural networks are typically stacked layers of interconnected neurons or nodes each having an activation function. The idea is to give our network several inputs (each input is a single feature). Then, we navigate through the network by applying weights to our inputs, summing them, and finally using an activation function to obtain the subsequent nodes. This process is repeated multiple times through the hidden layers, before applying a final activation function to obtain the output layer, our prediction. An example with n input nodes and 1 output node is shown below, with the step activation function : $f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$



The training takes place on the weights, which are initialized randomly. They are updated by trying to minimize training error using gradient descent, and are used after the training phase to make predictions.

Gradient Descent. Let's say we have an example of a regression problem, so for given input x we want to predict the output. We present the hypothesis as:

$h_\theta(x) = \theta_0 + \theta_1 * x$, where θ_0 and θ_1 are the parameters.

Then we need a function for minimization these parameters over our dataset. One common function that is often used is mean squared error, which measure the difference between the estimator (the dataset) and the estimated value (the prediction). It looks like this:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

It turns out we can adjust the equation a little to

make the calculation down the track a little more simple. We end up with:

$$\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

I will refer to the cost function as J(θ). In our problem we will need to minimize a cost function, called Mean Squared Error (MSE). The use of Gradient Descent function is valuable for minimizing it, by changing the theta values, or parameters, step per step, until we hopefully achieved a minimum.

We start by initializing theta0 and theta1 to any two values, say 0 for both, and go from there. Formally, the algorithm is as follows:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_i}J(\theta_0,\theta_1) \quad (\text{for } j=0 \text{ and } j=1)$$

and repeat until convergence:

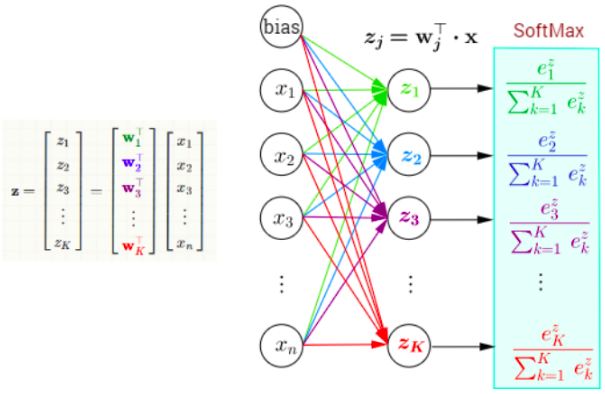
$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$
$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}$$

where α is the learning rate, or how quickly we want to move towards the minimum.

In our case, we construct a probability distribution of the 16 genres by running the outputs through a softmax activation function.

Simply speaking, the sigmoid function (check $ 3.7) only handle two classes, which is not what we expect. The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1.

The output of the softmax function is equivalent to a categorical probability distribution, it



tells you the probability that any of the classes are true.

Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, ..., K.

Defining the softmax as

$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^K \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

The model was compiled with Adam optimizer, since its is appropriate for problems with very noisy/or sparse gradients, and a categorical cross-entropy loss function, since we have multiple classes, and trained in batches of 150 instances for 50 iterations.

For the fully connected neural networks architecture we used:

- A Dense input layer of 150 neurons and rectifier linear unit activation function

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0. \end{cases}$$

- A Dense hidden layer of 100 neurons and rectifier linear unit activation function.

- A Dense output layer of 16 output classes and softmax activation function.

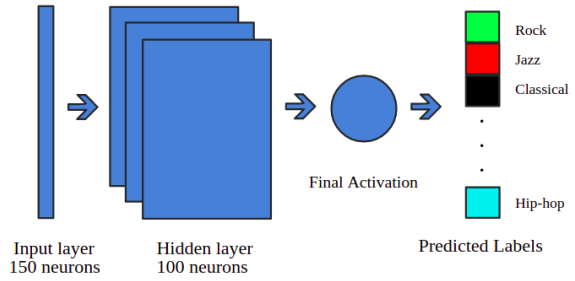


Figure 2. Neural Network

To create our fully-connected neural network, we begin with creating a Sequential model, a linear stack of 3 layers (input,hidden and output as mentioned above). Before training the model, we need to configure the learning process, which is done via the compile method. And then we train our model based on our training data, so as to learn our weights.

```
model = Sequential()
model.add(Dense(150, input_dim=train_x.shape[1], activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['mse','accuracy'])
model.fit(train_x, train_y, epochs= 50, batch_size=150 , shuffle=False, verbose = 1)
```

Figure 3. MLP model using Keras, Tensorflows high-level API

After trying various numbers of hidden units in the hidden layer, we conclude that the best accuracy is achieved with 140 neurons. However, we choose a hidden layer of 100 neurons since in that situation we have almost the same accuracy result and smaller execution time.

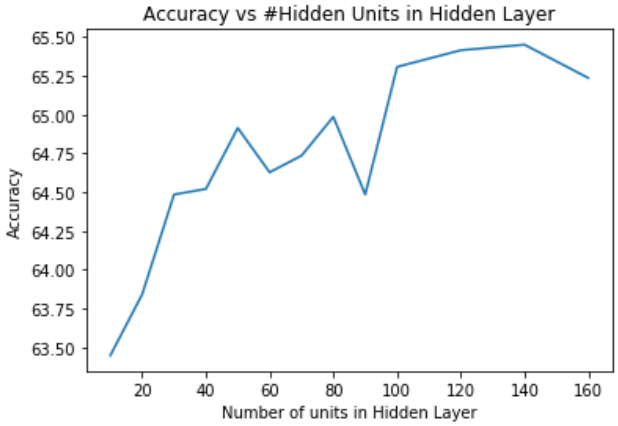


Figure 4. Accuracy vs Hidden Units in Hidden Layer

3.3. Support Vector Machine

In machine learning, Support Vector Machines are supervised learning models used for data analysis and pattern recognition in classification and regression problems. SVMs transforms the input feature space into higher-dimensional feature space using the kernel trick dot product, where (x,y) is replaced with k(x,y) = <f(x),f(y)> (f is called kernel function). We can find each dataset's sample distance to a given dividing hyperplane. We call margin the minimum distance from the samples to the hyperplane. The transformed data can be separated using a hyperplane, the dividing curve between distinct classes. The optimal hyperplane maximizes the margin.

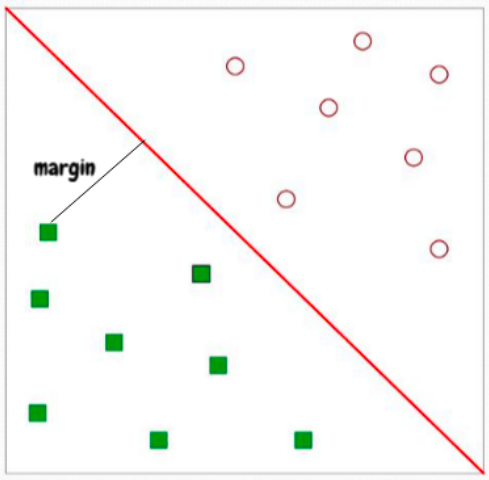


Figure 5. Hyperplane in 2d feature space

In other words, an SVM is a linear separator that focuses on creating a hyperplane with the largest possible margin. Its goal is to classifies a new sample by simply computing the distance from the hyperplane. On a two dimensional feature space, the hyperplane is a single line dividing the two classes (as we can see on figure 6). On a multi-dimensional feature space, where the data are non linearly separable an SVM cannot linearly classify the data. In this case it uses the kernel trick. The main concept has to do we the fact that the new multidimenional feature space could have a linear decisiong boundary which might not be linear in the original feature space. This can be done using various kernel functions such as :

- Polynomial kernel,8 popular in image processing. Equation is : $k(x, y) = (x \cdot y+1)^d$, where d is the degree of the polynomial
- Gaussian kernel, used when there is no prior knowledge about the data. Equation is :

$$k(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right)$$

, where σ is standard deviation.

- Gaussian radial basis function (RBF), also used when there is no prior knowledge about the data. Equation is : $k(x, y) = exp(-\gamma||x-y||^2)$, where $\gamma = 1/2\sigma^2$

- Hyperbolic tangent kernel, which can be used in neural networks. Equation is : $k(x, y) = tanh(kx \cdot y + c)$, for some k,c > 0

In our problem, a radial basis function (RBF) kernel is used to train the SVM because such a kernel would be required to address this nonlinear problem.

Large value of parameter C should cause a small margin, and the opposite. Small value of paramter C should cause a large margin. There is no rule to choose a C value, it totally depends on our testing data. The only way is to try various different values of the parameters and go with the those that gives the highest classification accuracy on the testing phase. As for the gamma parameter, if its value is low then even the far away points can be taken into consideration when drawing the decision boundary and the opposite. With a high gamma parameter, the hyperplane is dependent on the very close points and ignores the ones that are far away from it. So, the best parameters we used in the Support Vector Machine technique, after testing several variants with trial-and-error approach, were 10 for the C value, and 1 for the gamma value.

```
C = 10
gamma = 1
classifier = SVC(kernel="rbf",C=C,gamma=gamma)
classifier.fit(train_x, train_y)
y_pred = classifier.predict(test_x)
```

Figure 6. SVM using Sklearns SVC classifier class

3.4. Random Forest

A decision tree is a graphical representation of data which applies a branching method to describe with illustrations any possible result of a decision. Each tree branch stands for a possible option that is available for taking a decision. The depth of the tree is extended as new decisions need to be made. So the decision tree as a tree structure, has few components; internal nodes for a test on an attribute, the branches that represents a test result and leaf nodes (terminal nodes) that contains information for the class labels. The outcome of a test -based on the node's attributes- creates a new branch, and as we go from root to the next node and so on we reach a leaf node, that belongs to a unique label (class).

As we can see in the figure below, we have to decide for example if a person is fit. Following a series of boolean tests we will end up on a "fit" or "Unfit!" leaf node, concluded for the fitness's level of a person.

Random Forest is a summation of Decision Trees. The general idea of this technique is that

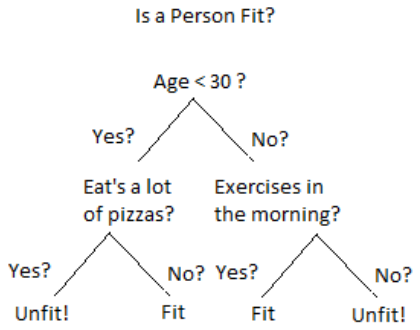


Figure 7. Decision tree toy example

a mixture of learning models raises the general result. Random forest builds multiple decision trees and merges them together to achieve the preciseness and stability of the prediction. In that way, it prevents overfitting by creating random subsets of the features, building smaller trees using these subsets and combines them so as to increase the overall performance. With overfitting we mean the result of an overly complex model with too many parameters. If a model is overfitted, we can result that it is inaccurate because its behaviour does not represents real data information.

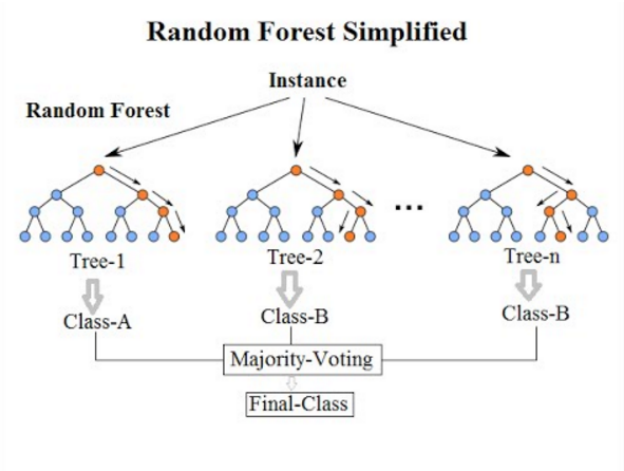


Figure 8. Random Forest Simplified

As we can see in the previous figure, a simplified Random Forest could categorize a sample to the class with the maximum ”votes” among each subtree.

Random Forest makes the model more random, while developing the trees. Rather than looking for the most significant feature while splitting a node, it scans for the best element among a random subset of features. This outcomes in a wide variety that by and large results in a greater model.

The parameters we used after trying several variants(check the figure below), were 29 estimators - the number of trees in the forest - and 29 as the depth of the tree. The algorithm provided us with a relatively high accuracy result in a short time, specifically 68.74% in 1.29 seconds. Generally our opinion was that the algorithm provided us with the best results in regard to how simple it was to develop and fast to execute.

```
clf = RandomForestClassifier(n_estimators=29, max_depth=29,random_state=1)
clf.fit(train_x,train_y)
y_pred = clf.predict(test_x)
```

Figure 9. RF using Sklearns RF classifier class

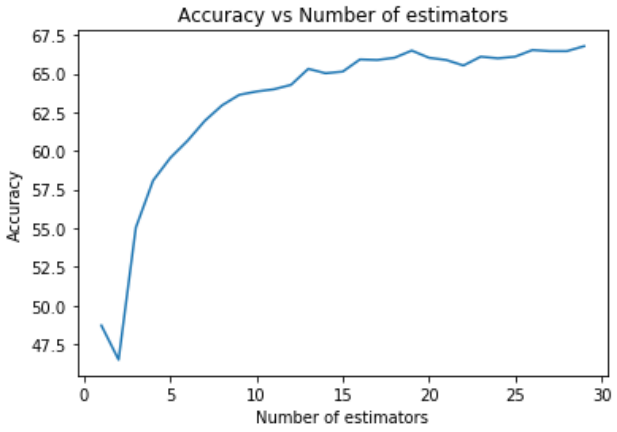


Figure 10. Accuracy vs Number of estimators

3.5. Gradient Boosting

Gradient boosting as a machine learning method for classification and regression problems is trying to create a model for predictions in the form of a mixture of inefficient prediction models, that are called decision trees. In the

boosting phase, every new tree is a fit on an altered version of the original data set. Firstly, the Gradient boosting trains a decision tree and assigns each observation an equal weight. After the first tree assessment, we lower the weights for the observations that are easy to classify and increase the weights of those that are hard to classify. Then, we grow the next tree on this weighted data where we try to improve the predictions of the first tree. Our new model is tree 1 together with tree 2. Then we compute the classification error from this combination of two-tree model and create a third one to predict the revised residuals. We repeat this process for a specified number of iterations. Subsequent trees help us to classify the non-well categorised observations by the former trees. The predictions of the final ensemble model is the weighted sum of the predictions made by the previous tree models.

After trying several variants, we set the values 0.1 and 150(with the smaller execution time) to the learning rate and the estimators - the number of boosting stages to perform - respectively and train our model with 74.21% accuracy, the highest between the eight algorithms. Though, Gradient Boosting takes 39.77 seconds to execute, much slower than the above-mentioned techniques.

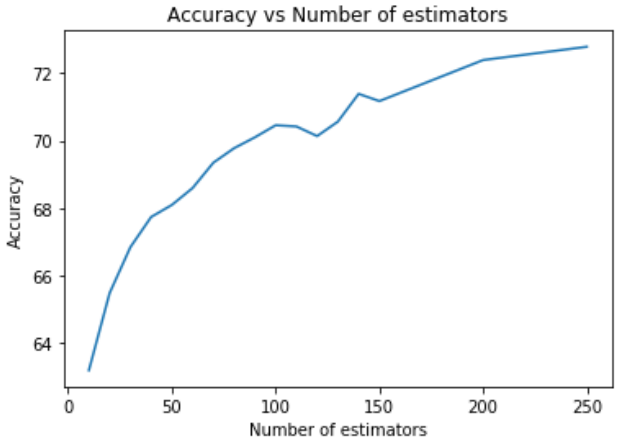


Figure 11. Accuracy vs Number of estimators

```
clf = GradientBoostingClassifier(learning_rate=0.1,n_estimators = 150,verbose=1)
clf.fit(train_x,train_y)
y_pred = clf.predict(test_x)
```

Figure 12. GB using Sklearns GB classifier class

3.6. K-Nearest Neighbors

The k-nearest neighbors (KNN) is a supervised machine learning algorithm useful for solving regression and classification problems in a simple and easy way. The KNN algorithm is based on the assumption that same things exist in a close area. In other words, similar things are close to one another.

KNN is based on the idea of similarity (also known as distance, proximity, or closeness) figuring the space between points on a graph. There are various methods of calculating distance, and one way might be preferable depending on the problem we are solving. For our problem, we use the most common distance metric, which is the Euclidean distance between two points x_1 and x_2, and is given by the formula:

$$d(x_1, x_2) = ||x_1 - x_2||_2$$

We will see now who does the KNN algorithm work. First of all, we have to load the data - usually called X- and their target values - usually called y- we want to classify. Then we initialize K to a preferable number of neighbors and for each data sample we compute the distance between the sample whose target value we want to classify and the current sample from the data. We add both the index and the distance of the query example to an ordered list of indices and distances and sort this list in an ascending order(from smaller to bigger), with the distance as order criteria. Finally, we pick the first K entries from the sorted list and get the labels of the selected K entries, so we can return the form of the K labels.

For example, as we can see in the figure below for K=3 the tested sample would have been classified to class B, and for K=7 would have been classified to class A.

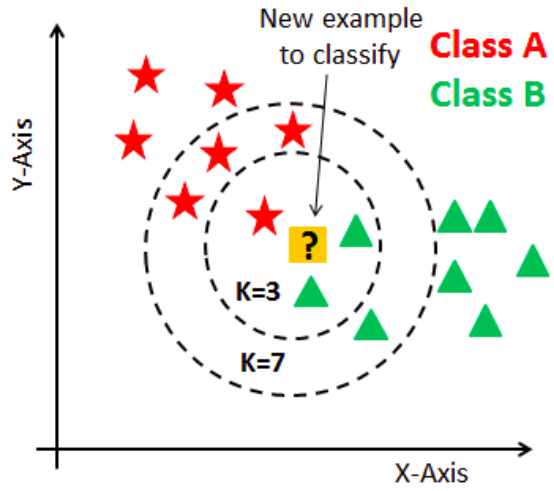


Figure 13. KNN example

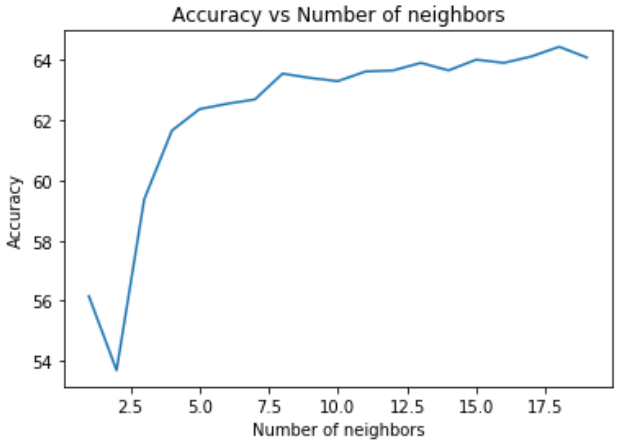


Figure 14. Accuracy vs Number of neighbors

```
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 15. KNN using Sklearns KNeighborsCLassifier class

In order to choose the best number for the variable K to suit with our problem, we have to execute the K-Nearest Neighbors algorithm some times with different values of K and choose the K that decreases the number of errors we meet while keeping up the ability of the KNN algorithm to make accurate predictions when it is provided with data that has never seen before. As the value of K is reduced to one, our predictions' stability is also decreased. On the other, as we raise K's value, our predictions' stability is increased to because of the majority voting in the compared samples. As a result, it is more possible to increase - until a specific level- the percentage of accuracy for our predictions.

In our problem, we choose 5 as the number of neighbors, despite the fact that we achieve the highest accuracy with 17, since we have better execution time.

3.7. Logistic Regression

Logistic regression is a supervised machine learning algorithm used for classification problems, and specifically for categorizing observations into a group of discrete classes. Although linear regression assign observations to a continuous number values, logistic regression applies on its output a tranformation - activation - function, called the logistic sigmoid function. In that wat it returns a probability value which can then be matched with two or more classes. Logistic Regression is used when the target - dependent - variable is categorical. For example, to predict whether an email is spam (1) or not (0) (binary logistic regression) or to predict whether a car with specific characteristics belongs to a model like BMW, Mercedes, Ford, etc (multiclass logistic regression).

Binary logistic regression. Suppose we are given a dataset that includes student exam results and we want to predict if a student will pass or fail the exam according to the combination of number of hours he has slept and hours spent for studying. In our data, we have to deal with the features of slept hours and studied hours and only two classes: passed (1) and failed (0), as a result of the pass or fail of a student's result.

Graphically we could represent these data with a scatter plot (Figure 17).

Sigmoid activation. In machine learning, we make use of the sigmoid function so as to match

Studied	Slept	Passed
4.85	9.63	1
8.62	3.23	0
5.43	8.23	1
9.21	6.34	0

Figure 16. The first four samples of these data

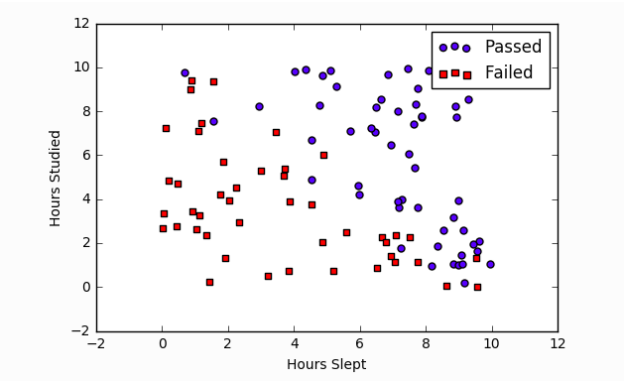


Figure 17. Scatter plot of our data

predicted values with the probabilities. The function convert any real value into a probability, which is in fact another value between 0 and 1.

- s(z) = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithms prediction e.g. ax + b)
- e = base of natural log

$$S(z) = \frac{1}{1 + e^{-z}}$$

Decision Boundary. The sigmoid function returns the predicted value of the probability in the range between 0 and 1. In order to map this to a

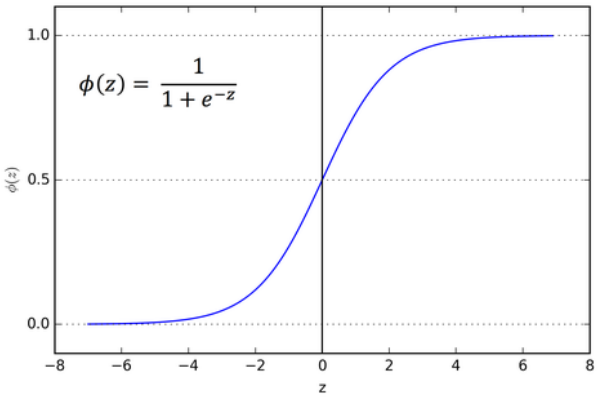


Figure 18. Sigmoid function

value that belongs to a discrete class (true/false, BMW/Ford/etc), we select a threshold value below which we will classify values into the second class (in the case of binary logistic regression) and above which we will classify values into the first class. The same concept is followed when we have more than two discrete classes (multiclass logistic regression).

$$p \geq 0.5, class = 1$$
$$p < 0.5, class = 0$$

For example, if our tipping point (threshold value) was 0.5 and our prediction function returned 0.8, we would classify this observation to the class 1 (positive value). If our prediction was 0.3 we would classify the observation to the class 2 (negative value). In the case of multiclass logistic regression where we have to decide between multiple classes to map the observation, we could map the observation to the class with the highest predicted probability.

We can create now a prediction function by taking into consideration our knowledge of decision boundaries and sigmoid functions. Such a function could compute the probability of our observation for being positive. In other words, the probability of our observation belongs to the class 1 and its notation is P(class=1). As the probability increases and reaches 1, our model be-

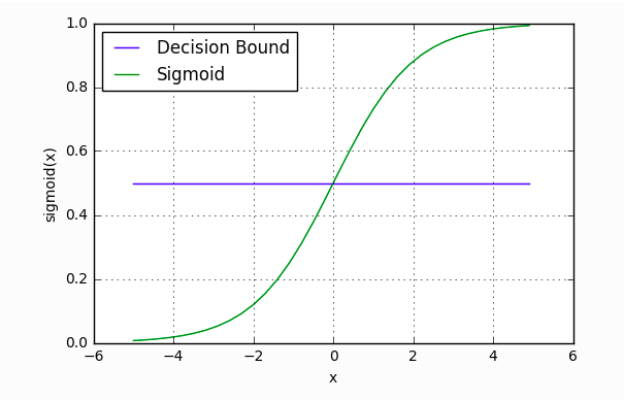


Figure 19. Decision boundary

comes more stable and we could be more confident that the observation is in class 1. On the other side, as the probability decreases and gets far away from 1, our model gives a higher probability that the observation is in class 2.

$$P(class = 1) = \frac{1}{1 + e^{-z}}$$ tr

So, now using the sigmoid function we will apply the transformation to our output and it will return a probability value between 0 and 1.
z = w_0 + w_1 Studied + w_2 Slept

The final step is to map the predicted probabilities to the discrete classes with labels 0 or 1. So, we classify our sample at class with label = 1 if the probability $\geq$ 0.5, else at class with label = 0.

In our FMA dataset, we use multiclass logistic regression since we have 16 discrete classes for our predicted labels. So we replace the definition of y = 0, 1 and use a new definition of multiple labels, y = 0, 1, ..., 15. In fact we run binary classification multiple times, once for each class. First, we divide the problem into n+1 (n=16) binary classification problems and for each class we predict the probability the observations are in that single class. The predicted class comes from the maximum probability between the classes.

In our problem, we use Sklearn's LogisticRegression classifier class, with lbfgs solver since we have a multiclass problem and can also handle multinomial loss. For the same reason multiclass parameter is equal to multinomial. In order for the method to converge we choose 300 as max iterations number.

```
clf = LogisticRegression(random_state=0, solver='lbfgs', max_iter=300,
                         multi_class='multinomial')
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 20. LR using Sklearns LogisticRegression classifier class

3.8. Linear Discriminant Analysis

For binary classification problems, we are familiar with using Logistic Regression algorithm. However, in a multiclass classification problem with more than two discrete classes, Logistic Regression is not the best algorithm due to its complexity. Instead, we prefer to use another dimensionality reduction technique: Linear Discriminant Analysis (LDA), useful for supervised machine learning problems.

Before we dig into the characteristics of Linear Discriminant Analysis, we will take a look of the Naive Bayes classification algorithm that creates the basic details for LDA.

Naive Bayes. Given a feature vector $X = (x_1, x_2, ..., x_k)$ and a class variable C_n, Naive Bayes Theorem mentions that:

$$P(C_n|X) = \frac{P(X|C_n)P(C_n)}{P(X)}, \ for\ n = 1, 2, ..., k.$$

$P(C_n|X)$ stands for the posterior probability, $P(X|C_n)$ for the likelihood, $P(C_n)$ for the prior probability of class, and P(X) for the prior probability of the predictor. We want to compute the posterior probability from the likelihood and prior probabilities.

Assuming the Naive independence, which claims that $P(X|C_n) = P(x_1, ..., x_k|C_n) = \prod_{i=1}^{k} P(x_i|C_n)$, the posterior probability can then be thought as:

$$P(C_n|X) = \frac{P(C_n)\prod_{i=1}^{k} P(x_i|C_n)}{P(X)}.$$

Then, the Naive Bayes classification problem turns into the statement: for different class values of C_n, find the maximum value of $P(C_n)\prod_(i = 1)^k P(x_i|C_n)$. This can be written as:

$$C = arg\max_{C_n} P(C_n)\prod_{i=1}^{k} P(x_i|C_n)$$

Discriminant analysis.
Now we can examine a wider concept, Discriminant analysis. Discriminant analysis can be analyzed by 5 steps:

1: Compute prior probabilities. The prior probability of class $P(C_n)$ is equals to the relative frequency of class C_n in the training set of our data.
2: Check the homogeneity of the samples' variances, actually check if K samples are from populations with equal covariance matrices. After this test we could decide which algorithm to use, Linear or Quadratic Discriminant Analysis.

- If there is variances' homogeneity, we should use Linear discriminant analysis: $\sum_1 = \sum_2 = ... = \sum_K = \sum$.

- If there is variances' heterogeneous, we should use Quadratic discriminant analysis: $\sum_i \neq \sum_j$ for some $i \neq j$.

3: Likelihoods' parameters estimation. We estimate the parameters (e.g.μ_i and $\sum$) of the conditional probability density functions $P(X|C_n)$ from the training set of our data. In fact, we have to take the assumption that the data are multivariate normally distributed.
4: Calculate discriminant functions. This computation will determine into which of the known populations to classify the new object.
5: Compute the performance of our classification: Make use of the cross-validation method in order to calculate misclassification probabilities. And we can now predict the new observations.

Linear Discriminant Analysis. We take the standard assumption that in population $_i$ the probability density function of x is multivariate normal with mean vector $_i$ and variance-covariance matrix (same for all populations). The equation to compute this normal probability density function is:

$P(X|p_i) = \frac{1}{(2\pi)^{p/2}|\sum|^{1/2}} exp[-0.5(X - \mu_i)'\sum^{-1}(X - \mu_i)]$

Now, with our knowledge about Naive Bayes classification algorithm states we are concerned about the fact that we make the classification for the example to the population for which $P(\pi_i)P(X|\pi_i)$ is maximized. To simplify our calculations, we apply to the previous equation a logarithmic transformation.

In Linear Discriminant Analysis, we take the decisions based on the **Linear Score Function**, which is defined as:

$s_i^L(X) = -0.5\mu_i'\sum^{-1}\mu_i + \mu_i'\sum^{-1}X + logP(\pi_i) = d_{i0} + \sum_{j=1}^{p} d_{ij}x_j + logP(\pi_i) = d_i^L(X) + logP(\pi_i)$

where $d_{i0} = -0.5\mu_i'\sum^{-1}\mu_i$ and d_{ij} = jth element of $\mu_i'\sum_{-1}$. And we call $d_i^L(X)$ the linear discriminant function.

From the above equation it is obvious that the right-hand expression seems like a linear regression with coefficients d_{ij} and intercept d_{i0}.

Given an example with a feature vector $X = (x_1, x_2, ..., x_p)$, we compute for each population the linear score function, and we map the given example into the population with the largest value of this score. In fact, we classify the example to the population that has the largest posterior probability of membership.

In our problem we use Sklearns LinearDiscriminantAnalysis classifier class to label our samples with a music genre.

```
clf = LinearDiscriminantAnalysis()
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 21. LDA using Sklearns LinearDiscriminantAnalysis classifier class

3.9. XGBoost

XGBoost belongs to the family of the ensemble learning methods. Some times, it could be insufficient to depend on the results of only one machine learning method applied to our data. Ensemble learning techniques use a systematic method to combine the predictive power of various learning methods. The output of this combination is a model that provides the totaled result from smaller-weaker- models. Most of the times, we use XGBoost algorithm with decision trees.

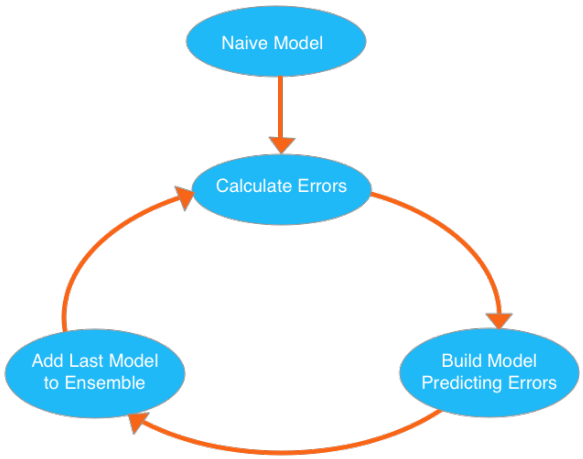


Figure 22. Gradient Boosted Trees diagram

XGBoost is the main model for dealing with the type of data you store in Pandas DataFrames, known as tabular data, and not other types of data like images and videos. XGBoost belongs to the Gradient Boosted Decision Trees algorithms, which we will examine later on.

The basic concept follow the circular rule where the method repeatedly creates new models and blends them into an ensemble model. At the beginning of this procedure, we calculate the errors for each observation in the data. Then we create a new model to predict those errors. We add the error-predicting results of this model to the total of the model. Then we add the predictions from all previous models in order to make a new prediction. We can use the added predictions to calculate new errors, create the next model, and add it to the ensemble.

In the boosting phase, each tree is built in a subsequent way, trying to reduce the residual errors of the previous trees. The learning procedure follows the previous tree and updates the remaining errors. The next growing tree will learn from the last version of the residual errors.

The boosting method in its basis has learners with high bias and low - bit higher than random predictions - predictive strength. Each of these base learners produce crucial information that contributes to the ensemble model for the predictive power. As a result, by combining these base-weak learners the overall boosting method develops strong predictive power and becomes a strong learner (with lower bias and variance).

In the boosting technique, trees with fewer splits are usually used, and as a result such small trees can me easily managed and interpeted. On the other side, the bagging techniques such as Random Forest examine trees that are extended to the maximum of their depth. So it is preferable to go with a boosting instead of a bagging algorithm, since it is an updated version of the last one. Also, we have to select optimally the stopping criteria since a huge number of trees can cause overfitting. Applying validation techniques like k-fold cross validation we can carefully choose parameters such as the depth of the trees, the number of splits per level, etc.

The boosting phase consists of three steps:

- An initial model F_0 is defined to predict the target variable y. This model will be associated with a residual $(y - F_0)$

- A new model h_1 is fit to the residuals from the

previous step

- Now, F_0 and h_1 are combined to give F_1, the boosted version of F_0. The mean squared error from F_1 will be lower than that from F_0:

$$F_1(x) < -F_0(x) + h_1(x)$$

To improve the performance of F_1, we could model after the residuals of F_1 and create a new model F_2:

$$F_2(x) < -F_2(x) + h_2(x)$$

This can be done for m iterations, until residuals have been minimized as much as possible:

$$F_m(x) < -F_{m-1}(x) + h_m(x)$$

Here, the additive learners do not disturb the functions created in the previous steps. Instead, they impart information of their own to bring down the errors.

As the first step, the model should be initialized with a function $F_0(x)$. $F_0(x)$ should be a function which minimizes the loss function or MSE (mean squared error), in this case:

$$F_0(x) = arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$
$$= arg\min_\gamma \sum_{i=1}^{n} (y_i - \gamma)^2$$

Taking the first differential of the above equation with respect to , it is seen that the function minimizes at the mean i=1nyin. So, the boosting model could be initiated with:

$$F_0(x) = \frac{\sum_{i=1}^{n} y_i}{n}$$

In our problem, we choose the default number of 100 estimators as it provides the highest accuracy in the shortest execution time.

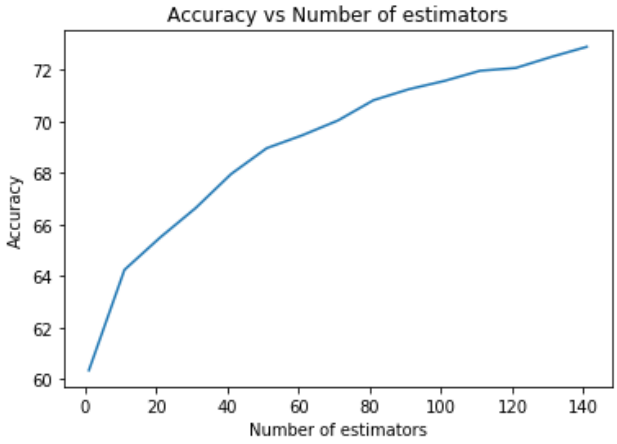


Figure 23. Accuracy vs Number of estimators

```
clf = XGBClassifier()
clf.fit(train_x, train_y)
y_pred = clf.predict(test_x)
```

Figure 24. XGB using Sklearns XGBclassifier class

3.10. Performance Comparison / Results

Our baseline was single-threaded CPU code. The libraries in python and sklearn are commonly used for machine learning purposes, so they are the de facto standard. The characteristics of our system are:

- Processor: Intel Core i7-4510U CPU @ 2.00GHz x4
- RAM: 8 Gb
- Cores: 4
- Sockets: 1
- OS type: Ubuntu 19.04

The quantitative metric which we used to judge our models is the combination of accuracy (that is, percentage of predicted labels which matched their true labels) and execution time. Below, we

provide an (accuracy,time)-based comparison between the used algorithms in order to give a better look into which algorithm is considered to be more suitable.

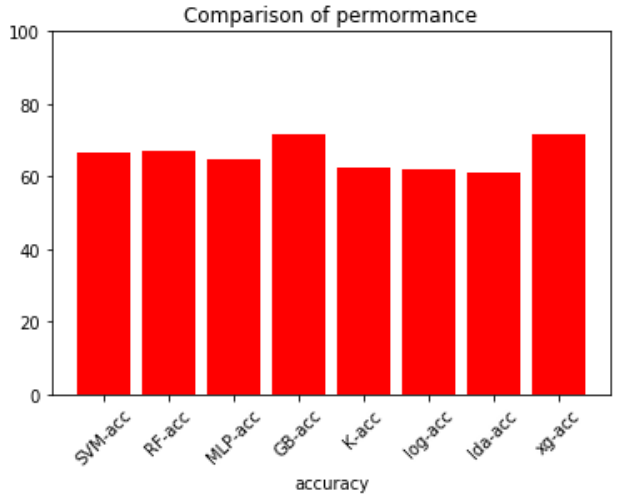


Figure 25. Accuracy comparison of the algorithms

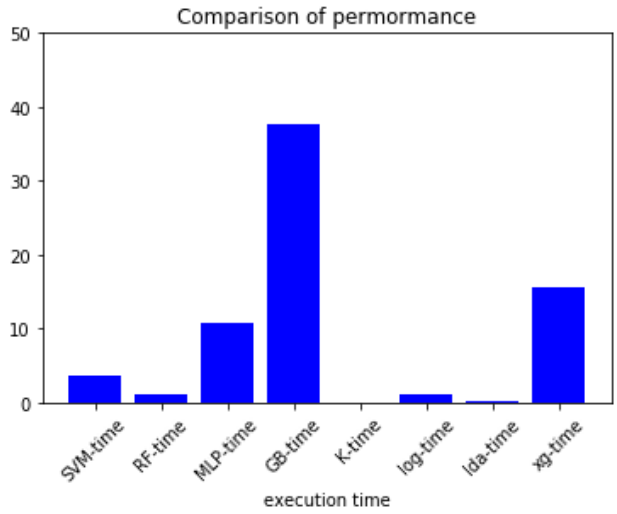


Figure 26. Execution time comparison of the algorithms

From the table below, it is obvious that Random Forest produces the better results combining the metrics of time and accuracy, and we also provide a bar plot to visually confirm that Random Forest performed better.

Another way of visualizing the performance of our best model is through a confusion matrix.

Table 2. Classification accuracies and execution times

ALGORITHM	ACCURACY	EXECUTION TIME
GB	74.21%	39.77s
MLP	65.14%	11.21s
SVM	66.74%	3.97s
RF	68.74%	1.29s
KNN	62.37%	0.03s
LR	61.87%	1.09s
LDA	61.12%	0.29s
XGB	71.53%	15.57s

A confusion matrix is a performance measurement for machine learning classification problem where output can be two or more classes. For a binary classification problem, it is a table with 4 different combinations of predicted and actual values.

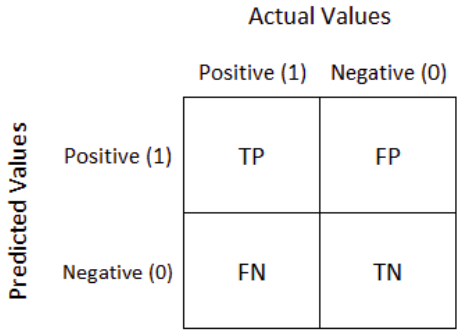


Figure 27. Confusion Matrix

- True positives (TP): In the example of predicting whether an email is spam (1) or (0), these are cases in which we predicted spam and it is also a spam email.
- True negatives (TN): We predicted non-spam, and it is not a spam email.
- False positives (FP): We predicted spam, but it is not spam.
- False negatives (FN): We predicted non-spam, but it is actually a spam one.

Here, as seen in the figure below Gradient Boosting algorithm did well with the rock, jazz

and instrumental genre.

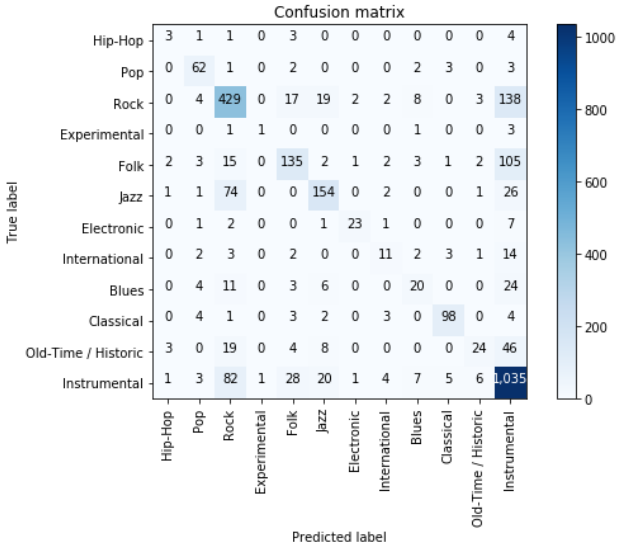


Figure 28. Confusion matrix for GB predictions

Additionally, it incorrectly classified some rock, as well as folk, as instrumental music. We should expect these misclassifications since rock is a genre that both encapsulates many different styles (hard rock, progressive rock, indie rock, alternativ e rock, etc.) and heavily influences many other derivative genres. Finally, only few samples can be labeled as Hip-hop or Experimental music - classes with the least importance - so it would not make any difference if we had not included them in our dataset.

4. Setup Installation

In order to test the algorithms over our dataset we need to install some tools, starting by updating the packages list and installing the prerequisites:

- *sudo apt update*
- *sudo apt install software-properties-common*
- *sudo apt-get -y install python3-pip*

Next, add the deadsnakes PPA to your sources list:

- *sudo add-apt-repository ppa:deadsnakes/ppa*

When prompted press Enter to continue.

Once the repository is enabled, install Python 3.7 with:

- *sudo apt install python3.7*

Then we will need to install the libraries below used by the project by entering in console the following command:

- *pip3 install pandas matplotlib keras scikit-learn numpy more-tertools seaborn xgboost*

Clone the repository locally by entering in console the following command:

- *git clone https://github.com/Kkalais/Music_Genre_Classification.git*

In order to run the code using the above-mentioned algorithms just enter in console the following commands :

- *python3 main.py name_of_the algorithm*

There is also a mode that runs all four algorithms consecutively, and produces a bar plot to compare the algorithms' results. Please enter in console:

- *python3 main.py comparative*

5. Future work

There are several directions that would be interesting to pursue in the future:

- Experiment with other types of deep learning methods - for example, Convolutional Neural Networks - that they will produce higher classification accuracies
- Try classifying music by another target-feature, e.g. by artist or by decade

- Including additional metadata text features such as album, song title, or lyrics could allow us to make an extension to music mood classification as well.

- If we are able to train a much deeper network that attains high accuracy, it would be interesting to plot the accuracy of a classical algorithm using different activation layers.

- Examine the application of Long Short Term Memory (LSTM) model in music genre classification

6. Acknowledgments

The project was implemented in the spectrum of a special issue course of the University of Thessaly, Electronic and Computer Engineering Department. The professor and supervisor of the project, prof. Yota Tsompanopoulou guided us to the majority of the knowledge, tools and skills needed for this purpose.

References

[1] Hareesh Bahuleyan, *Music Genre Classification using Machine Learning Techniques*, University of Waterloo, ON, Canada, 2018

[2] Danny Diekreoger, *Can Song Lyrics Predict Genre?*, Stanford University, 2012

[3] Michal Defferrardy, Kirell Benziy, Pierre Vandergheynsty, Xavier Bresson, *Fma: a dataset for music analysis*, LTS2, EPFL, Switzerland / SCSE, NTU, Singapore, 2017

[4] Abhishek Sen, BTech (Electronics), *Automatic Music Clustering using Audio Attributes*, Mumbai, India, International Journal of Computer Science Engineering (IJCSE)

[5] Matthew Creme, Charles Burlin, Raphael Lenain, *Music Genre Classification*, Stanford University, December 15, 2016

[6] George Tzanetakis, Student Member, IEEE, Perry Cook, Member, IEEE, *Musical Genre Classification of Audio Signals*,

[7] Carlos N. Silla Jr, Alessandro L. Koerich, Celso A. A. Kaestner, *A Machine Learning Approach to Automatic Music Genre Classification*, University of Kent Computing Laboratory, Pontifical Catholic University of Paran, Federal University of Technology of Paran

[8] Haojun Li (haojun), Siqi Xue (sxue5), Jialun Zhang (jzhang07), *Combining CNN and Classical Algorithms for Music Genre Classification*, Department of CS, Stanford University / ICME, Stanford University

[9] Muhammad Asim Ali, Zain Ahmed Siddiqui, *Automatic Music Genres Classification using Machine Learning*, Department of Computer Science, Karachi, Pakistan

[10] Kyuwon Kim, Wonjin Yun, Rick Kim, *Clustering Music by Genres Using Supervised and Unsupervised Algorithms*

[11] Anand Venkatesan, Arjun Parthipan, Lakshmi Manoharan, *Rock or not? This sure does. [Category] Audio Music*

[12] Derek A. Huang, Arianna A. Serafini, Eli J. Pugh, *Music Genre Classification*, Stanford University

[13] John Thickstun, Zaid Harchaoui, Sham M. Kakade *LEARNING FEATURES OF MUSIC FROM SCRATCH*, Department of Computer Science and Engineering, Department of Statistics, University of Washington, Seattle, WA 98195, USA

[14] Tiago Filipe Beato Mourato de Matos, *Statistical Models in Music Genre Classification*, Tcnico Lisboa, UL, Lisboa, Portugal

[15] Sergio Oramas, Oriol Nieto, Francesco Barbieri, Xavier Serra1, *MULTI-LABEL MUSIC GENRE CLASSIFICATION FROM AUDIO, TEXT, AND IMAGES USING DEEP FEATURES*, Music Technology Group, Universitat Pompeu Fabra / Pandora Media Inc. / TALN Group, Universitat Pompeu Fabra

[16] Roberto Basili, Alfredo Serafini, Armando Stellato, *CLASSIFICATION OF MUSICAL GENRE: A MACHINE LEARNING APPROACH*, University of Rome Tor Vergata, Department of Computer Science, Systems and Production, 00133 Roma (Italy)

[17] Ajay Prasadh Viswanathan, Sriram Sundaraj, *Music Genre Classification*, Department Of Computer Science and Engineering,National Institute of Technology, Trichy, India

[18] Bruna D. Wundervald, Walmes M. Zeviani, *MACHINE LEARNING AND CHORD BASED FEATURE ENGINEERING FOR GENRE PREDICTION IN POPULAR BRAZILIAN MUSIC*, Department of Statistics/Hamilton Institute, Maynooth University / Department of Statistics, Paran Federal University, February 12, 2019

[19] Michal Defferrard, Sharada P. Mohanty, Sean F. Carroll, Marcel Salath, *Learning to Recognize Musical Genre from Audio*, EPFL, Lausanne, Switzerland, 13 Mar 2018

[20] Chun Pui Tang, Ka Long Chui, Ying Kin Yu, Zhiliang Zeng, and Kin Hong Wong, *Music genre classification using a hierarchical long short term memory (LSTM) model*, Department of Computer Science and Engineering, The Chinese University of Hong Kong

[21] Benjamin Murauer, Gnther Specht, *Detecting Music Genre Using Extreme Gradient Boosting*, Universitt Innsbruck, Innsbruck, Austria

[22] Ritesh Ajoodha, Richard Klein, Benjamin Rosman, *Single-labelled Music Genre Classification Using Content-Based Features*,

[23] Kai Chen, Sheng Gao, Yongwei Zhu, Qibin Sun, *MUSIC GENRES CLASSIFICATION USING TEXT CATEGORIZATION METHOD*, Institute for Infocomm Research, 21 Heng Mui Keng, Singapore

[24] Cory McKay, *Using Neural Networks for Musical Genre Classification*, Faculty of Music, McGill University, Canada

[25] Anders Meng, Peter Ahrendt, Jan Larsen, IMPROVING MUSIC GENRE CLASSIFICATION BY SHORT-TIME FEATURE INTEGRATION, Informatics and Mathematical Modelling, Technical University of Denmark / Richard Petersens Plads, Building 321, DK-2800 Kongens Lyngby, Denmark