Stockholms universitet

Classification of Music Genres with eXtreme Gradient Boosting

Jesper Muren

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2019:8
Matematisk statistik
Juni 2019

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2019:8**
http://www.math.su.se

Classification of Music Genres with eXtreme Gradient Boosting

Jesper Muren*

June 2019

Abstract

The constant growth of the market for music streaming services such as Spotify, Pandora or iTunes is accompanied by a growth in the science of *Music Information Retrieval* or MIR. One big part of MIR is the classification of music, a topic that this thesis will cover. More specifically the classification of songs according to genres will be performed using various characteristics for songs obtained through Spotify. For this thesis the genres we will try to classify are Electro, Hip Hop, Jazz, Pop, R&B and Rock. The purpose of the thesis is to see if we can create a satisfactory classifier for our 6 genres as well as gain insight into how the classifier performs on different genres and with different features. The large quantities of data available when it comes to music has made various machine learning algorithms the natural choice for this task. For this thesis we will limit ourselves to using the relatively new and popular eXtreme Gradient Boosting or XGboosting. XGboosting is a version of gradient boosting which uses an ensemble of decision trees to turn many weak learners into one strong learner. It can be used for classification as well as regression problems and has shown promising performance for both. After tuning the parameters for our XGboost model an overall accuracy of 73.43% was obtained for our classifier, with significantly better performance on Electro, Jazz and Rock compared to R&B and Pop. Important features include for example speechiness for Hip Hop, acousticness for Jazz and danceability for Rock. The result is considered satisfactory when taking into account inherent difficulties in the task at hand as well as the somewhat lacking data set. The largest flaws of the data set is the high percentage of missing data for the lyrics as well as the limited amount of observations.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: jespermuren@gmail.com. Supervisor: Ola Hössjer.

Acknowledgements

This is a Bachelor's thesis of 15 ECTS in Mathematical Statistics at the Department of Mathematics at Stockholm University. I would like to thank my supervisor Ola Hössjer for his support and advice during the writing of this thesis.

Contents

1 Introduction

With the evergrowing music industry there is also growth in the science of getting information from music, often refered to as *Music Information Retrieval* or MIR. With the large amount of music available on music streaming services such as Spotify, Pandora or iTunes the need for efficient ways of categorizing music is growing, not only for the individual with a desire to categorize music but also for the companies when introducing recommendation features. One way to categorize music is by genre which is what we will explore in this thesis. In search of a way to automatize this task machine learning algorithms have become the natural choice. Supervised learning models such as support-vector machines, neural networks, random forest and tree boosting algorithms are all seeing use, for example Bahuleyan used these and other machine learning techniques for classifying music genres [2]. Because of the limited scope of this thesis, an interest in algorithms using decision trees and widespread positive results in the use of XGboosting [5] this algorithm was used for classification in this thesis. Extreme gradient boosting or XGboost is a version of gradient boosted decision trees which use an ensemble of weak learners (decision trees) to create one strong learner. The name extreme gradient boosting actually comes from the engineering goal of increasing the computational resources for these kinds of algorithms. This is a big reason for the popularity of XGBoost but it will not be the main focus of this thesis. For our purposes the main difference is the added regularization that XGboost provides compared to regular gradient boosting which is used to keep the model from overfitting the training data. All of this will be discussed in more detail in Section 2. We will use data obtained mainly from Spotify through their web API [13] and limiting ourselves to the genres Electro, Hip Hop, Jazz, Pop, R&B and Rock. When discussing the results we will touch on some of the inherent difficulties in classifying music according to genres that were encountered in this work, such as the subjectivity of the task.

1.1 Aim

The main aim of this thesis is to see if we can create a classifier with satisfactory accuracy using eXtreme Gradient Boosting for our data set. We would also like to gain understanding into the difference in performance on individual genres, which features are important for the model and how they effect it.

1.2 Disposition

From here we will continue the thesis by, in Section 2, going through the theory for the methods and concepts used in the thesis. Following that, in Section 3, we will introduce the reader to the data used for modeling and testing as well as discussing decisions and assumptions made for the data. In Section 4 we cover the modeling, which in the case of XGboosting consists mainly of the tuning of hyper parameters. We continue to Section 5 where the results of the final model are presented and these results as well as potential improvements are then

discussed in Section 6. Finally, Section 7 contains the summed up conclusion of the thesis.

2 Theory

This section contains the theory for methods and concepts used in the analysis conducted for this thesis. With few stated exceptions the theory described in the following pages will follow that of the book The Elements of Statistical Learning by Hastie, Tibshirani, Friedman (2017) [7]. Inspiration has been taken, especially regarding mathematical notation and layout, from Living with Trees - Predicting Swedish Apartment Prices with eXtreme Gradient Boosting by Hörnqvist (2019) [9].

2.1 Cross-validation

The theory in this section follows that of Section 7.10 in Hastie et al (2017) [7].

Cross-validation is one of the most commonly used methods to estimate the error in predicting response Y with model $\hat{f}(X)$ where X contains all predictor variables. With this method we directly get the extra-sample error which is the expected value of the loss of predicting Y with $\hat{f}(X)$

$$Err = E[L(Y, \hat{f}(X))].$$

This is done by using one part of the data to fit a model and another part to test said model. Doing this in several rounds with K different parts of the data is called K-fold cross-validation and will be the method used in this thesis.

2.1.1 K-fold cross-validation

In K-fold cross-validation we split the data into K equal parts where we for $k = 1, 2, \ldots, K$ in step k use the k:th part of the data as a validation set. This means that we use the other $K - 1$ parts of the data to fit a model and then use the k:th part to test this model. We do this using each different part of the data as validation set and the others to fit data and combine our K estimates of the prediction error.

This can be written more mathematically using the indexing function $\kappa : \{1, \ldots, N\} \mapsto \{1, \ldots, K\}$ that indexes each observation i, $i = 1, \ldots, N$ to fold k, preferably by randomization. We then get the estimate of the error by

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i))$$

where $f^{-k}(x)$ is the fitted function obtained when not using the k:th part of the data, y_i and x_i are the response and predictor variables for observation i.

Cross-validation can also be used to find optimal estimates of tuning parameters. Given a set of models $f(x, \alpha)$ with tuning parameter α we can with these models define

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha))$$

and find the estimate $\hat{\alpha}$ which minimizes this function and then choose $f(x, \hat{\alpha})$ as our final model.

2.2 Decision trees

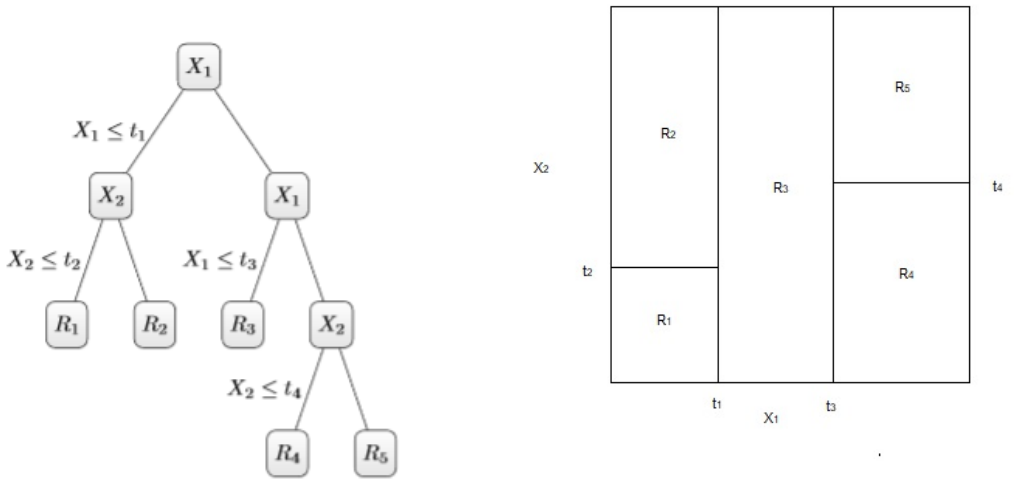
The theory in this section follows that of Section 9.2 in Hastie et al (2017) [7].

Decision trees is a powerful yet easily interpreted method applicable to supervised learning problems. Supervised learning relies on having labeled training data where each observation, $i = 1, 2, \ldots, N$, has p inputs and a response, (x_i, y_i) where $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. In supervised learning you use this training data to try and gain understanding of the problem you want to solve. Problems that supervised learning is used on is usually regression problems or classification problems, and since this thesis is one on multi-class classification we will put more focus on this.

There are several different decision tree algorithms such as CART, C4.5 and ID3, in this thesis we will be using and therefore only describing the method known as CART, *classification and regression trees*. As the name implies these trees can be used both for classification and regression.

These methods work by splitting the space of explanatory variables X into rectangles by recursively splitting the space parallel to the coordinate axes and in each rectangle fitting a simple model, like a constant, to all observation contained in it. We illustrate this in two different ways in Figure 1 with an example using two continuous explanatory variables X_1 and X_2 and response variable Y which could either be continuous or categorical. The right figure illustrates this in the previously explained way and the left figure shows how it can be represented in tree-form. In this example we begin by splitting the space at $X_1 = t_1$, after this we proceed by splitting subsequent sub-spaces, $X_1 \leq t_1$ is split at $X_2 = t_2$ whereas $X_1 > t_1$ is split at $X_1 = t_3$ and finally $X_1 > t_3$ is split at $X_2 \leq t_4$. This gives us five disjunct regions $R_1, R_2, \ldots, R_5$ also denoted as terminal nodes or leaves where all observations in region R_m, $m = 1, \ldots, 5$ are predicted with a constant γ_m. This constant in chosen in different ways depending on whether the response variable is continuous or categorical. For a continuous response variable γ_m is often chosen as the average of all observations y_i in region R_m and for a categorical response it is often chosen as the proportion of a class in the region, which can then by some threshold be converted to categories such as the majority class in the region.

Using the this and the indicator function notation, a decision tree can be expressed more compactly as



(a) Tree visualization

(b) Two-dimensional space splitting visualization

Figure 1: Two ways to visualize decision trees with 2 explanatory variables.

$$f(X) = \sum_{m=1}^{M} \gamma_m \mathbb{1}\{X \in R_m\}$$

for a tree with M terminal nodes.

The question of how to grow the tree remains, in other words how do we chose the variable X_j and split point s for each split. If we consider the first split of all our data the regions obtained by splitting variable X_j at s are:

$$R_1(j,s) = \{X|X_j \leq s\} \quad \text{and} \quad R_2(j,s) = \{X|X_j > s\}.$$

In each split like this j and s are chosen in a greedy manner in the sense that the split that minimizes a given criteria in each step is selected, without taking other steps into account. The criteria used for splitting in regression trees is the sum of squares $\sum(y_i - f(x_i))^2$ where the optimal $\hat{\gamma}_m$ is, as mentioned, the average of all y_i in region R_m. This means that for the split defined above we are looking for j and s which solve

$$\min_{j,s} \Big[\min_{\gamma_1} \sum_{x_i \in R_1(j,s)} (y_i - \gamma_1)^2 + \min_{\gamma_2} \sum_{x_i \in R_2(j,s)} (y_i - \gamma_2)^2 \Big]$$

and the inner minimization is solved by

$$\hat{\gamma}_1 = \text{ave}(y_i|x_i \in R_1(j,s)) \quad \text{and} \quad \hat{\gamma}_2 = \text{ave}(y_i|x_i \in R_2(j,s))$$

for any j and s.

The process of splitting is done in the same way for classification trees except that for classification we need some other criteria to be minimized. For K classes, N_m observations in region R_m, and the proportion of class k in node m defined as

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}\{y_i = k\}$$

the two most common criteria used are the following: The Gini index given by

$$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

and the Cross-entropy(deviance) given by

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}).$$

These two measures are prefered over the misclassification rate, which might seem like the natural choice. It is given by $1 - \hat{p}_{mk(m)}$, where $\hat{p}_{mk(m)}$ is the proportion of the majority class in node m. In other words the proportion of observations in a node not belonging to the majority class.

The reason why the Gini index and Cross-entropy are prefered even though all three measures are similar is that the former two are differentiable which we will see in coming sections is needed when we regard the minimization of these as numerical optimization. When considering these measures for two class classification we can see that all measures are minimized when the proportions of all classes within a leaf are 0 or 1, in other words when a leaf contains only observations from 1 class.

In theory we could continue doing splits of the feature space until we obtain perfectly homogeneous leaves like this but this could lead to overfitting our model to the training data making its performance worse on other or future data. On the other hand if we don't grow the tree large enough we might miss important structure in the data. This problem is in more general terms known as the Bias-Variance tradeoff which is often illustrated as in Figure 2 where there is high bias, low variance to the left. Since the model is simple it is underfitting and misses important structures in data. To the right there is high variance and low bias, since the model is overfitting to the training data it does not generalize well to the test data.

The most common way of dealing with this problem in decision trees is to chose a minimum number of observations a leaf must contain and stop growing the tree when this is reached, usually chosen depending on your data set. After this stop criteria is reached a method often called *pruning* is used where a cost complexity criteria is used to reduce the tree by balancing the complexity of the tree with the goodness of fit. For our use of decision trees in gradient boosting

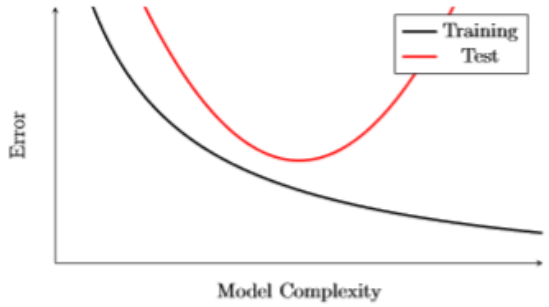


Figure 2: Illustration of training and test error when varying model complextiy from low to high.

the trees are by design kept small and their size will instead be decided using cross-validation. We will therefor not go into more detail on the concept of *pruning*.

2.3 Boosting

The theory in this section follows that of Section 10.1-3 and 10.9 in Hastie et al (2017) [7].

Boosting is an ensemble method that combines many weak learners to create one strong learner, a weak learner is one that is only slightly better than chance. In boosting these weak learners are trained in sequence on modified versions of the data, where each added learner tries to correct the mistakes of previous learners and it gives a weight, usually depending on how well it performs on the training data. The base weak learners are then collected to make a committee of weighted votes to choose the final prediction.

A common choice of weak learners in boosting are decision trees used as basis functions in an additive expansion. This is known as Forward Stagewise Additive Modeling and will be described below.

2.3.1 Forward Stagewise Additive Modeling

The basis function expansion as mentioned earlier can be written on the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \theta_m)$$

where β_m $m = 1, 2, \ldots, M$ are the weights for the individual basis functions and $b(x; \theta)$ are the basis functions with explanatory variables x and parameters θ. Since the basis functions in our case are decision trees $\{\theta_m\}_{m=1}^{M}$ contains the split variables, split points and leaf predictions.

The usual way of fitting these kind of models is to minimize some loss function $L(y, f(x))$ averaged over the training data

$$\min_{\{\beta_m,\theta_m\}_1^M} \sum_{i=1}^{N} L\Big(y_i, \sum_{m=1}^{M} \beta_m b(x;\theta_m)\Big).$$

Minimizations like this are often computationally intensive so forward stagewise additive modeling approximates the solution to this minimization by sequentially adding new basis function with weights that minimize the loss in the current step, without changing previous parameters and weights. In this way the problem is reduced to solving for a basis function $b(x;\theta_m)$ and weight β_m to add to current expansion f_{m-1}. This is illustrated in Algorithm 1 in pseudo-code.

Algorithm 1 Forward Stagewise Additive Modeling algorithm

1: Initialize $f_0(x) = 0$
2: **for** $m = 1$ to M **do**
3: Compute :

$$(\beta_m, \theta_m) = \arg\min_{\beta,\theta} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\theta)).$$

4: Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\theta_m)$
5: **end for**
6: **return** $\hat{f}(x) = f_M(x)$

2.3.2 Boosted Trees

For future reference we rewrite our earlier representation of decision trees in the following way

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}\{X \in R_j\}$$

where $\Theta = \{R_j, \gamma_j\}_1^J$. If we use these as the basis function in forward stagewise additive modeling, the sum of such trees make a boosted tree model

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$$

where, as in the third step of Algorithm 1, in each step we must solve the minimization

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)). \tag{1}$$

Here $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ has the region set and leaf constants for the next tree, given all previous trees contained in current model $f_{m-1}(x)$.

The hard part is usually finding the regions R_{jm}. Given that we know these solving for γ_{jm},

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}), \tag{2}$$

is often easy as the optimal $\hat{\gamma}_{jm}$ is usually the average of all y_i in R_{jm} for regression or the modal class for classification.

As finding the regions is a harder task approximate solutions are often used. In addition to choosing regions by splitting in a greedy fashion by only considering minimization of the loss criteria in the current step without taking other steps into consideration it may also be favourable to use a more smooth and convenient loss criterion

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)).$$

An example of this was mentioned in Section 2.2 where Gini index or Cross-entropy is often used rather than the misclassification rate.

After finding an approximate solution this way and using $\hat{R}_{jm} = \tilde{R}_{jm}$ we can proceed with the easier task of estimating γ_{jm} with the original criterion.

2.4 Gradient Boosting

The theory in this section follows that of Section 10.10 in Hastie et al (2017) [7]. Gradient boosting uses the idea that boosting can be seen as a numerical optimization problem of some differentiable loss function $L(f)$. Minimization of the loss of predicting y_i with $f(x)$ through

$$L(\mathbf{f}) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

is given by

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}),$$

with $\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T$ being the values of the prediction at each data point.

Numerical optimization solves this minimization problem with a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \quad \mathbf{h}_m \in \mathbb{R}^N$$

where after the first guess $\mathbf{f}_0 = \mathbf{h}_0$, subsequent $\mathbf{f}_m$ are given by $\mathbf{f}_{m-1} + \mathbf{h}_m$ where $\mathbf{h}_m$ tries to improve on what the model performs poorly at in previous steps.

2.4.1 Steepest Descent

A way of choosing the component vector $\mathbf{h}_m$ is by realizing that a function decreases most rapidly in the direction of the negative gradient $\mathbf{g}_m$ which has the components

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}.$$

Steepest descent therefore uses $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ where ρ_m is a constant that controls the length of the step taken in the direction of the negative gradient as to not make a too long or short of a step. This length is found through line search as the solution to

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m).$$

2.4.2 Gradient boosted trees

The way of using the concept explained above with decision trees is by realizing that just like forward stagewise boosting, steepest descent is a greedy strategy that in every step minimizes some loss criterion without taking other steps into consideration. This means that the tree predictions $T(x_i; \Theta_m)$ added to minimize (1) are analogous to the components of the negative gradient.

Since our goal is to make a model applicable to unseen data and the gradient of the loss function is only defined for the data points x_i in our training data we instead use trees $T(x; \Theta)$ whose predictions are as close to the negative gradient as possible. With squared error as a measure of closeness we get

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2.$$

When the tree that minimizes the above expression with regions $\tilde{R}_{jm}$ is constructed we can easily find the leaf weights with (2) as shown in Section 2.3.2. In other words this is not the tree added as the step from f_{m-1} to f_m, it is the tree that provides the regions which lets us easily compute the optimal weights.

Since the chosen loss function can differ between problems and applications there are several different gradients that can be used for this. In this thesis we will be using the multinomial deviance, or log loss which is the one used when the response variable takes values in the unordered set $G = \{G_1, \ldots, G_K\}$, when performing K-class classification in XGboost. This is given by

$$L(y, p(x)) = -\sum_{k=1}^{K} y_k \log(p_k(x)) \tag{3}$$

where $y = (y_1, \ldots, y_K)$ and y_k takes the value 1 if it belongs to class G_k and 0 otherwise and $p_k(x) = \mathrm{P}(Y_k = G_k|x)$ where $p_k(x)$ is given by the softmax function

$$p_k(x) = \frac{\exp(f_k(x))}{\sum_{l=1}^{K} \exp(f_l(x))} \tag{4}$$

which makes sure that $0 \leq p_k(x) \leq 1$ and that the class probabilities $p_l(x)$, $l = 1, \ldots, K$ sum to one.

Using (4) we see that the log loss (3) can be rewritten as

$$L(y, p(x)) = -\sum_{k=1}^{K} y_k f_k(x) + \log\left(\sum_{l=1}^{K} \exp(f_l(x))\right).$$

From this we get that the components of the negative gradient for step m, $m = 1, \ldots, M$ and tree k, $k = 1, \ldots, K$ is given by

$$-g_{ikm} = \left[\frac{\partial L(y_i, f_1(x_i), \ldots, f_K(x_i))}{\partial f_k(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} = y_{ik} - p_k(x_i)$$

for $i = 1, 2, \ldots, N$.

Using this we present the Gradient Tree boosting algorithm for K-class classification in pseudo code in Algorithm 2. Since there are K classes, each with its own regression function f_k, we fit K regression trees with associated regions R_{jkm} and coefficients γ_{jkm} in each iteration, one for each class.

Algorithm 2 Gradient Tree boosting algorithm for K-class classification

Initialize $f_{k0}(x) = 0, \quad k = 1, 2, \ldots K.$
for $m = 1$ to M **do**
  Set :

$$p_k(x) = \frac{\exp(f_{k,m-1}(x))}{\sum_{l=1}^{K} \exp(f_{l,m-1}(x))}, \quad k = 1, 2, \ldots, K.$$

  for $k = 1$ to K **do**
    (i)Compute : $-g_{ikm} = y_{ik} - p_k(x_i), \quad i = 1, 2, \ldots, N.$
    (ii)Fit a regression tree to $-g_{ikm}, \quad i = 1, 2, \ldots, N.$
    (iii)With obtained regions $R_{jkm}, \quad j = 1, 2, \ldots J_m,$ Compute:

$$\gamma_{jkm} = \arg\min_{\gamma} \sum_{x_i \in R_{jkm}} L(y_i, f_{k,m-1}(x_i) + \gamma), \quad j = 1, 2, \ldots J_m.$$

    (iv)Update: $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} \mathbb{1}(x \in R_{jkm})$.
  end for
end for
return $\hat{f}_k(x) = f_{kM}(x), \quad k = 1, 2, \ldots, K.$

2.5 Tree boosting parameters

The theory in this section follows that of Section 10.11-12 in Hastie et al (2017) [7].

Boosted trees have 3 main hyper parameters which will be discussed in this section, tree size - J, number of boosting iterations - M and learning rate - η. We will begin by discussing J - the size or number of terminal nodes in each tree.

2.5.1 Tree size J

This parameter is one used instead of the pruning we mentioned in Section 2.2. Pruning is the natural choice when a single tree is the whole model but since boosting fits trees additively this is a poor method as pruning each tree in the model would increase computation time significantly. The solution is to set a maximum number of terminal nodes for all trees, $J_m = J \ \forall m$. This J should be chosen depending on data at hand but Hastie et al (2017) [7] mention that values in the range $4 \leq J \leq 8$ yield good results from experience. Therefore, values in this interval can be tested to find which one performs best on the validation set. The tree size J also dictates the number of interaction effects, since only 1 variable is used in each split a tree of depth J can only have a maximum of $J - 1$ interactions between variables.

2.5.2 Regularization M and η

The other two parameters M and η are used for regularization. Since each boosting iteration adds a tree (or K coupled trees for K-class classification) which gives the model a better fit on the training data, a large M would seemingly be prefered but this can lead to overfitting the model to the training data, leading to worse performance on unseen data as shown in Figure 2. A suitable number of iterations that gives a good fit but no excessive overfitting is often obtained through cross-validation. Finally we can also use η for regularization, the learning rate - η is a shrinkage technique where $0 \leq \eta \leq 1$ is used to scale the contribution of each added tree, η would be added to line 8 in Algorithm 2 in the following way

$$f_{km}(x) = f_{k,m-1}(x) + \eta \sum_{j}^{J_m} \gamma_{jkm} \mathbb{1}(x \in R_{jkm}).$$

Since a smaller learning rate gives each tree less of an effect increasing η has a similar effect as decreasing M, it will give a higher training risk. This means that at a desired level of training risk reducing the learning rate has to be compensated by an increase of the number of iterations M. Smaller values of the learning rate have empirically been shown to perform best Friedman (2001) [6], often ($\eta < 0.1$). The accompanied increase in computations brought by having to increase M is often acceptable in part due to the previously mentioned maximum tree depth, keeping the trees small and therefore less complicated.

2.6 XGboost

In this section we will cover the details of eXtreme gradient boosting, more commonly refered to as XGboosting, the subsequent theory will follow that of the paper XGBoost: A Scalable Tree Boosting System by Chen Guestrin (2016) [5], the developers of XGboost.

XGboosting is similar to that of the gradient boosting we described in Section 2.4 although it uses a slightly different approach than the one described in that section. For starters it uses the Newton method to approximate the loss function in each step with a second order Taylor expansion. As a reminder to the reader the Taylor theorem tells us that a linear approximation of $f(x)$ around a can be given by

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2.$$

Given a twice differentiable convex loss function L at step m, with XGboost we want to minimize following expression at step m

$$\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)). \tag{5}$$

Here using our above notation we can see $f(x)$ as the loss function we want to approximate, $f(a)$ as the loss function in the previous step f_{m-1} and $(x-a)$ as the tree $T(x_i; \Theta_m)$ we want to add in this step. If we use the following more convenient notation

$$g_m(x_i) = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} \quad \text{and} \quad h_m(x_i) = \frac{\partial^2 L(y_i, f_{m-1}(x_i))}{\partial^2 f_{m-1}(x_i)}$$

we can, using the second order Taylor expansion, approximate equation (5) well with

$$\sum_{i=1}^{N} \Big[L(y_i, f_{m-1}(x_i)) + g_m(x_i)T(x_i; \Theta_m) + \frac{1}{2}h_m(x_i)T(x_i; \Theta_m)^2 \Big]$$

and removing the constant terms we get the following expression to be minimized

$$\sum_{i=1}^{N} \Big[g_m(x_i)T(x_i; \Theta_m) + \frac{1}{2}h_m(x_i)T(x_i; \Theta_m)^2 \Big]. \tag{6}$$

In reality XGboost uses an objective function made up out of not only a loss function but also a regularization term Ω, this regularization term is given by

$$\Omega(T) = \Gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \gamma_j^2, \tag{7}$$

where, as a reminder, J is the number of terminal nodes in the tree and γ_j the coefficients for these nodes.

From the XGboost documentaion [15] it can be found that there is also an option to increase this regularization term to include $\alpha \sum_{j=1}^{J} |\gamma_j|$. An in-depth discussion on this addition to the regularization term will not be included in this thesis so α will be set to zero. The reason for this and further discussions around the parameters of the regularization term will be discussed later in Section 2.7.

If we return to equation (6) and include the regularization term Ω we can write equation (6) as

$$\sum_{i=1}^{N} \Big[g_m(x_i) T(x_i; \Theta_m) + \frac{1}{2} h_m(x_i) T(x_i; \Theta_m)^2 \Big] + \Omega(\Theta_m).$$

If we rewrite this with the further expanded notation for trees $T(x_i; \Theta_m) = \sum_{j=1}^{J} \gamma_{jm} \mathbb{1}\{x_i \in R_{jm}\}$ and use the fact that regions R_{jm} are disjoint we get

$$\sum_{i=1}^{N} \Big[g_m(x_i) \sum_{j=1}^{J} \gamma_{jm} \mathbb{1}\{x_i \in R_{jm}\} + \frac{1}{2} h_m(x_i) \sum_{j=1}^{J} \gamma_{jm}^2 \mathbb{1}\{x_i \in R_{jm}\}^2 \Big] + \Omega(\Theta_m).$$

Then using that all terms where $x_i \notin R_{jm}$ are zero and expanding $\Omega(\Theta_m)$ we can rewrite this again as

$$\sum_{j=1}^{J} \Big[\Big(\sum_{x_i \in R_{jm}} g_m(x_i) \Big) \gamma_{jm} + \frac{1}{2} \Big(\sum_{x_i \in R_{jm}} h_m(x_i) \Big) \gamma_{jm}^2 \Big] + \Gamma J + \frac{1}{2} \lambda \sum_{j=1}^{J} \gamma_{jm}^2$$

and by moving the last sum in we get

$$\sum_{j=1}^{J} \Big[\Big(\sum_{x_i \in R_{jm}} g_m(x_i) \Big) \gamma_{jm} + \frac{1}{2} \Big(\sum_{x_i \in R_{jm}} h_m(x_i) + \lambda \Big) \gamma_{jm}^2 \Big] + \Gamma J.$$

Finally we simplify the notation again with $G_{jm} = \sum_{x_i \in R_{jm}} g_m(x_i)$ and $H_{jm} = \sum_{x_i \in R_{jm}} h_m(x_i)$ and get the final expression

$$\sum_{j=1}^{J} \Big[G_{jm} \gamma_{jm} + \frac{1}{2} \Big(H_{jm} + \lambda \Big) \gamma_{jm}^2 \Big] + \Gamma J \tag{8}$$

to be minimized at step m.

If we now consider a given tree structure as we have done previously when computing leaf weights and differentiate (8) with respect to γ_{jm} and set to zero, we find the optimal leaf weights to be

$$\tilde{\gamma}_{jm} = -\frac{G_{jm}}{H_{jm} + \lambda}. \tag{9}$$

Then, by substituting this expression into (8) we get the optimal value of the loss function for a given tree structure as

$$-\frac{1}{2}\sum_{j=1}^{J}\frac{G_{jm}^2}{H_{jm}+\lambda}+\Gamma J,$$

and this equation is used to score tree structures in order to choose optimal splits.

With this we get a formula for how to choose between potential splits in a greedy manner by from the top of the tree and downwards finding the split that maximizes the *Gain* in each leaf. Consider letting $G_L^2/(H_L+\lambda)$ and $G_R^2/(H_R+\lambda)$ score the left and right leaf respectively after a split and $G^2/(H+\lambda) = (G_L^2 + G_R^2)/(H_R+H_L+\lambda)$ be the score of the unsplit leaf we get following loss reduction to maximize at each split

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{G^2}{H+\lambda}\right]-\Gamma \tag{10}$$

where a split is made only if $Gain > 0$ which is why Γ is often refered to as minimum split loss.

Using this and extending to the K-class classification case we present the XGboost algorithm in pseudo code in algorithm 3.

Algorithm 3 XGboosting algorithm for K-class classification

Initialize $f_{k0}(x) = 0, \quad k = 1, 2, \dots K.$
for $m = 1$ to M **do**
  for $k = 1$ to K **do**
    (i)Compute : $g_{km}(x_i) = \frac{\partial L(y_i, f_{k,m-1}(x_i))}{\partial f_{k,m-1}(x_i)}, \quad i = 1, 2, \dots, N.$
    (ii)Compute : $h_{km}(x_i) = \frac{\partial^2 L(y_i, f_{k,m-1}(x_i))}{\partial^2 f_{k,m-1}(x_i)}, \quad i = 1, 2, \dots, N.$
    (iii)Fit a tree $T(x_i, \Theta_{km}) \quad i = 1, 2, \dots, N$ with splits that maximize:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{G^2}{H+\lambda}\right]-\Gamma$$

    (iv)With obtained regions from optimal splits, compute:

$$\tilde{\gamma}_{jkm} = -\frac{G_{jkm}}{H_{jkm}+\lambda}, \quad j = 1, 2, \dots J_m.$$

    (v)Update: $f_{km}(x) = f_{k,m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{jkm} \mathbb{1}(x \in R_{jkm})$.
  end for
end for
return $\hat{f}_k(x) = f_{kM}(x), \quad k = 1, 2, \dots, K.$

2.6.1 Sparsity-aware split finding

More often than not data sets for real-world problems contain features with missing values or a large number of zero-valued observations. In order for the model to take this into account XGboost adds a default direction for such values to go to in each split. This default direction is found by splitting based on the available data for the split variable and then computing the *Gain* for both cases of letting missing data go to the left or right node. Whichever direction minimizes the loss is chosen as the default direction.

2.7 XGboost parameters

Other than the tree boosting parameters we discussed in Section 2.5 XGboost has a plethora of parameters of its own, both for trees and data subsampling. For this thesis we will limit ourselves to looking at and tuning the ones explained in this section due to time and complexity restrictions. We will begin this section by discussing the tree parameters we encountered in Section 2.6.

2.7.1 Tree parameters

We will start by addressing the part we excluded from the regularization term as mentioned in previous section.

2.7.1.1 α - L1 regularization

As a reminder the part we chose to exclude from the regularization term was $\alpha \sum_{j=1}^{J} |\gamma_j|$. This term was excluded for two reasons, the first being that L1 regularization, analogous to Lasso regression is not included in the paper [5] that this section is based on. The second reason is that L1 regularization is often used to aid in feature selection by shrinking features to zero if they are unimportant in order to improve speed of the algorithm once implemented. As the number of features in our data set is not very large this is not a priority.

2.7.1.2 λ - L2 regularization

The parameter λ performs L2-regularization, analogous to ridge regression. We can see in (9) and (10) that λ has an effect on the leaf weights as well as the *Gain* and therefore the structure of the tree. This regularization term will have a larger effect on nodes with a smaller second derivative (Hessian), hence shrinking the leaf weights more as well as negatively effecting the gain score.

Interested reader can find more about lasso and ridge regression in Section 3.4 of Hastie et al (20017) [7].

2.7.1.3 Γ - Minimum split loss

This parameter controls the reduction in loss required to split, and as mentioned before a split is only made if the gain exceeds zero. As we can see in (10) this

is when the loss reduction is higher than Γ. This is another way to control size of the tree.

2.7.1.4 h_{min}- Minimum child weight

The last tree parameter we will cover in this thesis is h_{min}, usually refered to as minimum child weight, where child is another word used for leaf. This parameter uses that the Hessian can be seen as an instance weight for an observation, if the sum of the Hessian of observations in a node does not exceed h_{min} the node will not be split and the tree will not be grown further. This is yet another way to control the size of the tree. To understand why the Hessian is used we can look at the case where log loss is used as loss function. The log loss as shown in (3) has the second order derivative $p_k(x)(1-p_k(x))$ when taken with respect to $f_k(x)$. Which as a reminder is related to $p_k(x)$ through (4). This means that when the node is very homogeneous (contains many correctly predicted observations) the sum of the Hessian will be small and further splitting may lead to overfitting.

2.7.2 Subsampling parameters

In this thesis we will consider column subsampling by tree as well as row subsampling, as a reminder the observations $i = 1, \ldots, N$ are organised as rows and the p predictors as columns. These take values in the range $(0, 1]$ and in each iteration XGboost randomly samples the chosen fraction of total columns/rows of data to grow the tree from. Subsampling is used to prevent overfitting by increasing variance between trees. This is achieved by growing them with different subsets of the data and thereby reducing the correlation between them.

2.8 Model interpretation

In this section we will briefly cover the confusion matrix which is a commonly used technique to display performance of a classifier as well as the concept of feature importance for boosted tree models.

2.8.1 Confusion matrix

As mentioned above confusion matrices are a good way to visualize the performance of your model. A confusion matrix is a matrix with predicted classes displayed along the rows and actual class labels displayed along the columns. In the confusion matrix it is easy to see, for a given class, all the correctly predicted observations which are called true positives (tp), all observation mistakenly predicted as the given class which are called false positives (fp), all the observations of the given class predicted as another which are called false negatives (fn) and all the observations not of the given class which are predicted as another class are called true negatives (tn). With these definitions metrics to evaluate our classifier can be calculated. A selection of such metrics, intended to be used

Table 1: Measures for classification

Measure	Formula	Description
Recall(Sensitivity)	$\frac{tp}{tp+fn}$	Rate of positives actually classified as positives
Specificity	$\frac{tn}{fp+tn}$	Rate of negatives actually classified as negatives
Precision	$\frac{tp}{tp+fp}$	Rate of actual positives out of all observations classified as positives
F1 Score	$\frac{2 \cdot tp}{2 \cdot tp+fn+fp}$	Performance metric based on both recall and precision
Accuracy	$\frac{tp+tn}{tp+fp+fn+tn}$	Overall effectiveness of classifier

in Section 5 with description, are shown in Table 1, see also Hossin, M. and Sulaiman, M. N. [8].

These metrics are natural for a two by two confusion matrix but they can also be extended to K-class classification in different ways. We will do this by considering a one versus all situation where one reduces the classification to a binary one by looking at one class at a time and consider that class versus all the rest.

2.8.2 Feature importance

The theory in this section follows that of Section 10.13 in Hastie et al (2017) [7]. As shown in the visualization of a decision tree in Figure 1, one of its strengths is its ease of interpretability. Unfortunately this is lost when our model no longer consists of only one tree but instead an ensemble of them. Since we are still interested in the relative importance or contribution of each feature in $X = (X_1, \ldots, X_p)$ we need an importance metric for our features. This was proposed for a single tree T by Breiman et al. (1984) [4] as

$$I_\ell^2(T) = \sum_{j=1}^{J-1} \hat{\imath}_j^2 \mathbb{I}(v(j) = \ell) \tag{11}$$

which is the measure of importance for feature X_ℓ for $\ell = 1, \ldots, p$. The idea is that the variable used to split a node is the one that gives the optimal improvement $\hat{\imath}_j^2$ in squared error risk. We use this and sum over the $J-1$ internal nodes and at each node j where feature X_ℓ is used to split the node $(v(j) = \ell)$ we add the squared improvements. This gives us the relative square importance for variable X_ℓ in the case of a single tree.

For an additive tree expansion we compute (11) for all trees and average them

$$\bar{I}_\ell^2 = \frac{1}{M}\sum_{m=1}^{M} I_\ell^2(T_m)$$

which gives us the relative square importance of variable X_ℓ for the entire additive tree expansion. This can be generalized to the K-class classification case as well where we have one separate model for each class, in this case we can find the overall importance of feature X_ℓ by averaging over all classes.

3 Data

In this section we will discuss and analyze the data used in this thesis, all data is obtained using the R package Spotifyr which is a wrapper for the Spotify web API [13] and also has functions to obtain lyrics for songs from the website Genius.com. The data gathered for this thesis contains 6847 observations, which was then split into training data and test data using a 80:20 split. When making models the goal is prediction of unseen data, because of this it is sometimes (often) good practice to obtain another set of data to test on after the model has been finalized. This is in part because often the things we wish to predict are time sensitive. For this thesis this was decided to be unnecessary. The reason is that even though there are differences over time in music, for example 1950's rock probably has some different characteristics from contemporary rock, our training data already spans a very large time period. The relative short time during which this thesis was written is also had an impact. Another reason is that the way our data was gathered makes it extremely difficult to obtain another reasonably sized data set to test on.

3.1 Predictor variables

We started by manipulating the lyrics and cleaning up the data obtained through Spotifyr by mostly removing duplicate observations and excluding variables included in the data not of interest to us such as the Spotify ID for the track and URL links to full audio analysis of tracks. After this data cleaning we ended up with the predictor variables shown and described in Table 2. Most descriptions were obtained from Spotify's audio feature value description [12].

3.2 Response variable

The response variable in our data is genre. The response contains six classes being the genres Rock, Pop, R&B, Electro, Hip hop and Jazz. We have limited our classification to one with these six classes for a few reasons including time consuming data collection and availability of data, which will be discussed further in Section 6. The proportion of classes in our data is shown in Table 3 and as we can see the classes are quite well balanced with Rock having slightly more songs than the others and R&B slightly less.

Table 2: Predictor variables with value ranges or categories and descriptions.

Variable	Value range or categories	Description
danceability	0.0 to 1.0	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.
energy	0.0 to 1.0	Energy represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy.
key	A, A#, B, C, C#, D, D#, E, F, F#, G, G#	The estimated overall key of the track.
loudness	-60 to 0.0	The average loudness of a track in decibels.
mode	major, minor	Mode indicates the modality of a track, the type of scale from which its melodic content is derived.
speechiness	0.0 to 1.0	Speechiness detects the presence of spoken words in a track.
acousticness	0.0 to 1.0	A confidence measure of whether the track is acoustic.
instrumentalness	0.0 to 1.0	Predicts whether a track contains no vocals.
liveness	0.0 to 1.0	Detects the presence of an audience in the recording.
valence	0.0 to 1.0	A measure describing the musical positiveness conveyed by a track.
tempo	0.0 to infinity	The overall estimated tempo of a track in beats per minute (BPM).
duration ms	0.0 to infinity	The duration of the track in milliseconds.
time signature	1 to 5	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats there are in each bar.
words	0 to infinity	Count of words in lyrics chosen based on number of occurrences in songs and genres. Words used are the following: "love, baby, feel, time, heart, girl, gonna, day, no, ooh, life, night, black, bad, hey, bitch, fuck, shit, nigga, niggas, yeah, up, ayy, it, fire, instrumental, dream, arms, eyes.

Table 3: Class proportions in our data set.

Class	Proportion
Electro	0.175
Hip Hop	0.165
Jazz	0.166
Pop	0.165
R&B	0.128
Rock	0.201

3.3 Data transformation

All predictor variables shown in Table 2 except the word count based on lyrics are obtained directly from Spotify's web API [13] and had a natural way to be quantified. The word count however had to be obtained through manipulation of the lyrics of each song, the method for this was done as follows: Using the part of the data set aside for training, as to not use the test to fit on itself, we find the five most commonly used words for each song. We collect all these words for songs in each genre and then find the ten most commonly used words in songs for each respective genre. We do this in 2 steps in order to avoid having songs that use certain words excessively to skew the results. After this we count the occurrences of each word in the lyrics of each song.

XGBoost requires categorical variables to be numerical so the two predictor variables that are categorical, key and mode are encoded using one-hot encoding. Xgboost also requires this for the response variables class labels. In our case they need to be numbered from 0 to 5 as we have 6 classes, numbered alphabetically with Electro as 0, Hip Hop as 1 and so forth.

3.4 Missing data

When considering how to deal with missing data you want to try and understand why the data is missing. The usual way to deal with missing data is to either discard the observations with missing data, discard the variables with missing data, try to impute them with for example the mean of the non missing values of the variable or have the algorithm deal with them.

When there is missing data in a data set the best situation is the one when the data is *missing completely at random* (MCAR). If we consider the variable with missing data Z and a set of observed variables $\boldsymbol{X}$ then we can express MCAR as follows

$$\mathrm{P}(Z \text{ is missing}|\boldsymbol{X}, Z) = \mathrm{P}(Z \text{ is missing})$$

in other words that Z being missing is independent of its own missing value as well as all the observed variables $\boldsymbol{X}$. This assumption can be hard to test for and is also often not entirely satisfied for real world data set. A weaker more

Table 4: The five pairwise most correlated predictor variables in the data set.

Variables	Correlation
loudness & energy	0.76
duration ms & instrumentalness	0.37
valence & danceability	0.31
valence & energy	0.26
speechiness & danceability	0.25

common assumption is that data is *missing at random* (MAR) which can be expressed as

$$\mathrm{P}(Z \text{ is missing}|\boldsymbol{X}, Z) = \mathrm{P}(Z \text{ is missing}|\boldsymbol{X})$$

or in other words Z being missing can depend on $\boldsymbol{X}$ but not on its own value. As you might expect you cannot really test for MAR so you have to make an assumption. If your data is assumed to be MAR it is often deemed ignoreable in real word data sets, Missing Data, Allison (2002) [1].

In our data set the only variables that contain missing data are the word counts. Since there is data missing from over 50% of the songs removing the effected observations is not considered an option. The assumption of MAR is deemed justified as we do not see any reason the missingness would depend on how many times a certain word appears in the lyrics. For this reason and because XGboost deals with missing values through it's sparsity-aware split finding discussed in Section 2.6.1, all the variables and observations are left in the data set for modeling.

3.5 Correlation

Decision trees by design handle collinearity quite well since given two correlated variables the tree will just choose one to split on and the other one will be given less importance. For this reason dimensionality reduction by removing collinear variables is not as important in models using decision trees. That said it is still good practice to remove perfectly correlated variables from data in order to, for instance, simplify computations. In Table 4 we see the five most correlated predictor variables in our data.

As we see in Table 4 there is quite a high correlation between loudness and energy but because the correlation is not perfect and the number of predictor variables in our data is already quite small we choose to keep all variables in the data.

4 Modeling

In this section we will cover how the final model for the classification is decided. Since XGboost is the chosen algorithm for this thesis modeling consists mainly

Table 5: Hyper-parameter tuning approach and suitable parameter ranges

Hyper-parameter	Tuning approach	Range
Boosting iterations(trees) M	Default tune $\rightarrow$ Fine Tune	100-1000
Max tree depth J	Grid Search	(4,6,8,10)
Learning rate η	Default tune $\rightarrow$ Fine Tune	$(2-10)/M$
Minimum split loss Γ	Fixed	0
L2 regularization λ	Grid Search	(0.01,0.1,1)
Column Sampling Proportion	Grid Search	(0.4,0.6,0.8,1)
Row Sampling Proportion	Grid Search	(0.5,0.75,1)
Minimum child weight h_{min}	Fixed $\rightarrow$ Fine Tune	(1,3,5,7)

of tuning the parameters for this algorithm. We will be using the programming language R for the modelling in this thesis but XGboost is, for example, also available in Python, Java and C++. As XGboost is still relatively new on the machine learning scene definite instructions on parameter tuning is not available. Therefore we will use suggestions for general tree boosting methods, such as the range for tree depth mentioned in Section 2.5.1 as well as value ranges of parameters shown to work well in practice by Zhang (2015) [16] and Thakur (2016) [14].

4.1 Parameter tuning

There are several ways to go about tuning hyper-parameters for machine learning problems, the two most commonly mentioned ones may be grid search and random search. In this thesis we will be using grid search as it appears to be the the most commonly used one. The interested reader is directed to: Random Search for Hyper-Parameter Optimization, Bergstra and Bengio (2012) [3] for an in depth look at random search for hyper-parameter tuning.

In grid search we create a grid of all possible combinations of parameter values and using cross validation on the training set we test these combinations for the one that gives best result on chosen metric, in our case minimization of the log loss. As expected this can lead to an extremely computationally heavy task as the number and range of hyper-parameter grows. To combat this we use value ranges for parameters shown to give good results as mentioned above, as well as splitting up the parameter tuning in parts. Ranges that have been shown to work well for XGboost are shown in Table 5 as well as how we split up the process of tuning.

As shown in Table 5 we begin by tuning a default model with the learning rate η and the number of boosting iterations M. At this stage we test a few relatively high values of η (0.1, 0.5, 1) and decide the value of M with cross validation, the high value of η makes sure the boosting rounds are kept low and following grid search kept at a reasonable level computationally. After the

Table 6: Tuned parameters of the final model

Parameter	Value
Boosting iterations M	1200
Max tree depth J	6
Learning rate η	0.01
Minimum split loss Γ	0
L2 regularization λ	1
Column sampling proportion by tree	0.6
Row sampling proportion by tree	0.75
Minimum child weight h_{min}	5

default tuning we perform grid search on the parameters λ, column sampling proportion by tree, row sampling proportion by tree and max tree depth. Finally after tuning these parameters we fine tune η, typically with much lower values (0.01 - 0.1) and the number of iterations M(therefore higher) as well as the minimum child weight h_{min} that we kept fixed in the previous steps. The value of the minimum split gain Γ is kept fix at 0 throughout. After completing these steps we have the finally tuned parameters for our model ready to be applied to the test set.

5 Results

This section will mainly contain the numerical results of running the part of the data set aside for testing through our classifier while a more in depth discussion of the results is left to Section 6.

5.1 Model parameters

After following the steps outlined in Section 4.1, using the part of our data set aside for training, the tuned parameters that decide our final model are the ones shown in Table 6.

5.2 Model overall results

As the main goal of the classification in our problem is to minimize the number of incorrect answers and our data is relatively well balanced an appropriate main metric to measure the performance of our model is the overall accuracy as described in Section 2.8.1. The accuracy score achieved with the final model on the test data was 73.43%. The final classifications are displayed in the confusion matrix of Figure 3.

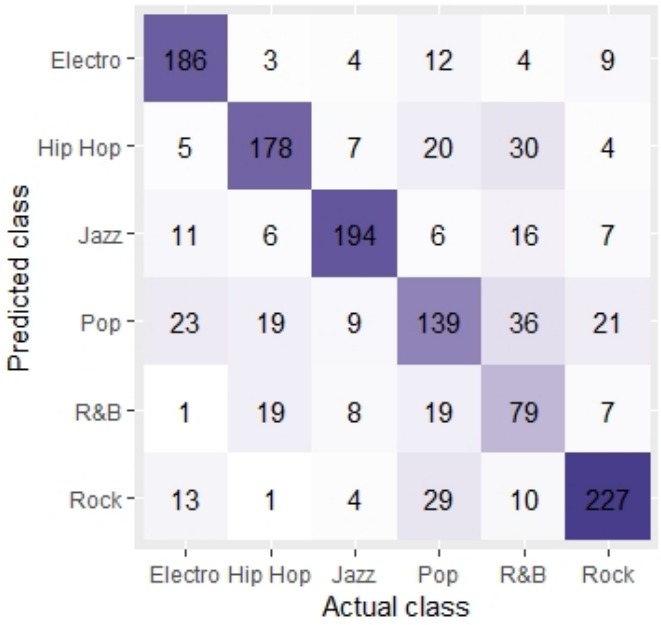


Figure 3: Confusion Matrix with the results of applying the classification method to test data.

5.3 Class specific results

The performance of our model on the individual classes is measured by extending the metrics from Section 2.8.1 to the multiclass case, by considering classification of each genre in turn versus a clumped category of all other genres. These class specific metrics, calculated from the confusion matrix, are shown in Table 7.

6 Discussion

In this section we will more thoroughly discuss the results presented in Section 5. We will also examine the feature importance for our model and finally discuss possible improvements to be made in future work.

6.1 Fitting

As can be seen in Table 6 the final parameters include changes from the default values in almost all of the regularization parameters. An explanation to this can be found by looking at Figure 4. We can see in the left panel depicting the log loss of 5-fold cross-validation of the default model for three different values of η that, especially for higher values of η, there's a significant upswing in the log loss curve. This indicates that the model is prone to overfitting to the

Table 7: Performance metrics calculated for each class.

Genre	Recall(Sensitivity)	Specificity	Precision	F1 Score
Electro	0.7782427	0.9716060	0.8532110	0.8140044
Hip Hop	0.7876106	0.9421053	0.7295082	0.7574468
Jazz	0.8584071	0.9596491	0.8083333	0.8326180
Pop	0.6177778	0.9053462	0.5627530	0.5889831
R&B	0.4514286	0.9546599	0.5939850	0.5129870
Rock	0.8254545	0.9477544	0.7992958	0.8121646

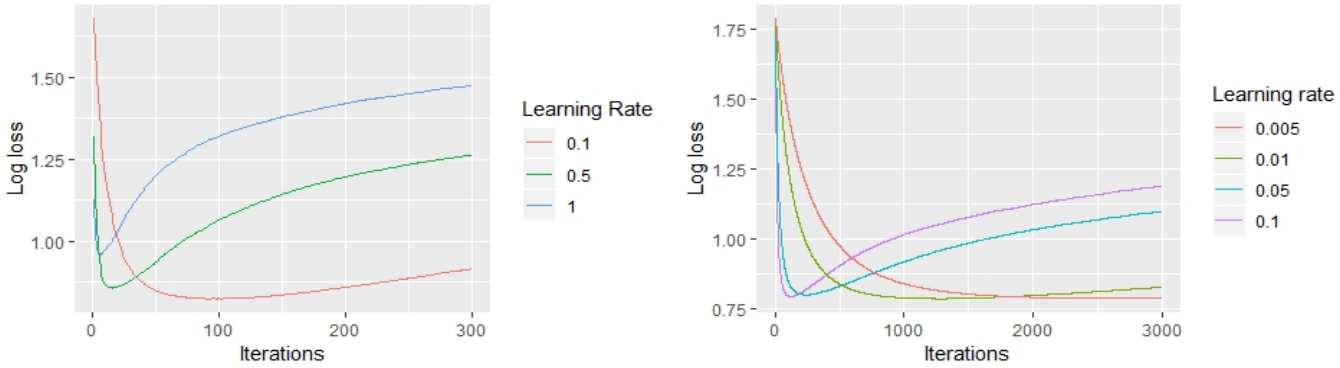


Figure 4: Log loss curves of 5-fold cross validation for different learning rate values when tuning the default model, showed in the left panel, and for the tuned model, showed in the right panel.

training set. The corresponding behaviour is also depicted in Figure 2. In order to lighten the computational load for the grid search, learning rates lower than 0.1 where not considered for the default model. As we can see the smallest log loss is obtained at roughly 100 iterations with learning rate 0.1 and therefore this value is chosen for the default model.

In the right panel of Figure 4 we find corresponding 5-fold cross-validation log loss curves for a selection of η-values for the tuned model. We see that the higher values of the learning rate still suffer quite severely from overfitting while lowering the learning rate levels out the curve. With the significantly different looking curves it is worth pointing out that all four learning rate values achieved very similar lowest log loss values but the lowest was achieved with a learning rate of 0.01. It is possible that a marginally lower log loss could be achieved by lowering the learning rate even more but this was not considered worthy the required increase of computational power. By tuning the model a roughly 4.9% decrease in cross-validated log loss was achieved.

6.2 Overall performance

As mentioned in Section 5 the overall performance of the classifier measured in accuracy is 73.43%. As good overall performance is one of the main aims with this thesis the obvious question is: Is this good? This is not an easy question to answer as it depends on several things, for example good in comparison to what and under what circumstances.

With a completely balanced data set the accuracy for random guessing would be $1/K$ for K-class classification, our data is close to being entirely balanced with Rock being the most prevalent class with 20% of the observations, so compared to random guessing our classifier performed quite well. On the other hand, if we imagine a real world implementation of this model in some application, would users be satisfied with this accuracy? Probably not.

This being said there are also some problems inherent to classifying music genres, for one there is some subjectivity in deciding music genres as there are often no clear definitions of genres and there can also be overlap, as for Pop-Rock, or a song can contain elements of more than one genre. Music is also often classified according to genre for artist or albums and not for songs. This is the case for Spotify which made data collection cumbersome and time consuming. All these things among other may lead to ambiguity in the labels of training data which of course will affect any supervised learning method, McKay & Fujinaga [11].

With all this in mind as well as the limited scope and time constraint for this thesis the performance of our model is deemed relatively good. Improvements and potential solutions for some of these problems will be further discussed in Section 6.5.

6.3 Classwise performance

Further analysis of how the model performs on the individual classes can aid in understanding the overall performance of the classifier by giving insight in where the model has a good performance and where it is not performing well. For this thesis we chose to use the F1 score as main metric of how well the classifier performs on individual classes. Another choice would be balanced accuracy which is usually a better choice with unbalanced data and when true negatives are of large importance. Since we are using the one versus all approach described in Section 2.8.1 the true negative count in our calculations become very large as they contain all observations correctly not classified as the class we are currently interested in, even if the observation is misclassified among other classes. This is the reason for the large specificity values for all classes in Table 7. They are inflated because of the large true negative counts. Because our data is quite well balanced and because of the specificity inflation we chose instead to use the F1 score which does not use this metric. Instead it uses recall and precision which do not include the true negative count.

As can be seen in Table 7, recall, precision and therefore the F1 score is very low for classes Pop and R&B compared to the other classes while Jazz, Rock and

Electro all receive an F1 score of over 0.8. The fact that the model is performing worse on Pop and R&B is also quite obvious by looking at the confusion matrix in Figure 3, where we can see that both classes are commonly misclassified with R&B being misclassified more often than not. This is somewhat expected for the genre Pop. A clue can perhaps be gleaned from its name as it comes from the phrase "Popular Music". A fair assumption is probably that the genre Pop generally is broader than other genres and also often contains elements from other genres. As for R&B it is less obvious why it would be harder to classify but from the confusion matrix we can see that it predominantly is missclassified as Pop or Hip Hop which from my experience are big components of contemporary R&B.

6.4 Feature Importance

As one of the aims of this thesis is to examine at which level different features affect the classification model as well as in what way they affect it we will in this section examine the feature importances in our model. The XGboost package provides three different ways of measuring feature importance, these are described below following XGboost documentation [15]

- Gain - A feature's relative contribution to the reduction of the loss function, this is XGboosts version of the feature importance metric described in Section 2.8.2.
- Cover - The relative quantity of observations related to a feature, calculated with the number of observations that go through splits made with a given feature.
- Frequency - The relative number of times a feature is used to split over all trees in model.

As one might suspect having three different measures of importance has the potential to cause confusion as each of them could indicate different features being the most important. It has also been shown that feature importance measures such as Gain and Frequency can sometimes be inconsistent and that there may be other more suitable measures. One such measure is the Shapley additive explanations value or SHAP value. That said, this is decided to be an extension of the analysis outside the scope of this thesis but the interested reader will find more about everything discussed in this paragraph in, Consistent Individualized Feature Attribution for Tree Ensembles Lundberg et al (2018) [10].

When considering feature importance in an XGboost model the general consensus is that the Gain usually is the best measure to use for feature importance. Frequency is generally not used as it is just a simpler way of measuring Gain.

In Figure 5 we can see the most important features for the overall classifier according to the three different metrics.

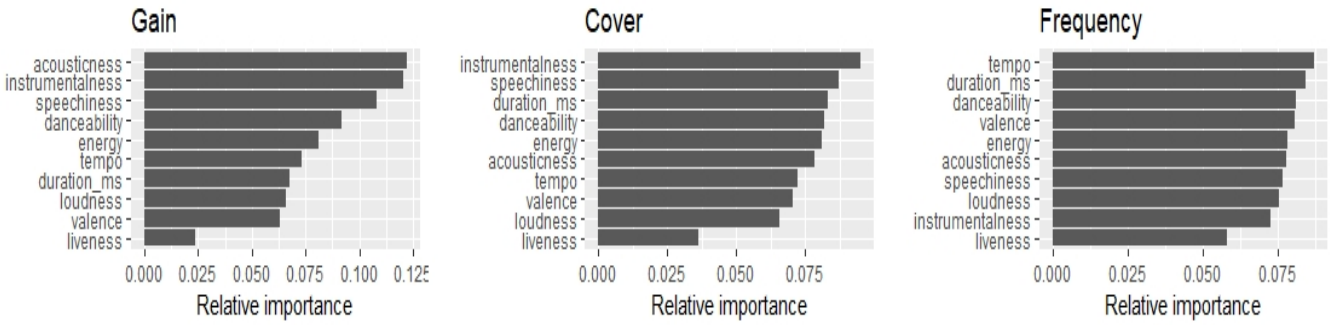


Figure 5: Feature importance for the chosen classifier. The ten most important features according to the measures Gain, Cover and Frequency are shown.

We can see that when considering the Gain as measure, acousticness and instrumentalness are the most important followed by speechiness and danceability. It is also noteworthy that for all measures the top 10 important features are variables directly from the Spotify data, none of the wordcounts are there. This is perhaps not entirely suprising as the word counts are essentially 1 variable "lyrics" divided into 29 features. This was done in an attempt to define explanatory variables that made more sense to use in the model. If we sum up the relative importance of all words using the Gain measure we receive a total value of roughly 0.159 which is even higher than for acousticness. It is not certain that this number is entirely trustworthy though as it might not fully make sense to sum the importance variables because of potential correlation. But it is still considered worth mentioning as to not forget the contribution of the lyrics to model performance.

6.4.1 Classwise feature importance

The feature importance presented in Figure 5 although important is not very intuitively interpretable. A perhaps better and certainly more intuitive way to consider the feature importance is to look at which feature is most important for the specific classes. As to not clutter this section with too many figures the plots of the three measures of feature importance for each class is shown in Figure 8 of the appendix. The most important features according to the Gain measure for our 6 classes are:

- Electro - tempo
- Hip Hop - speechiness
- Jazz - acousticness
- Pop - intrumentalness
- R&B - energy

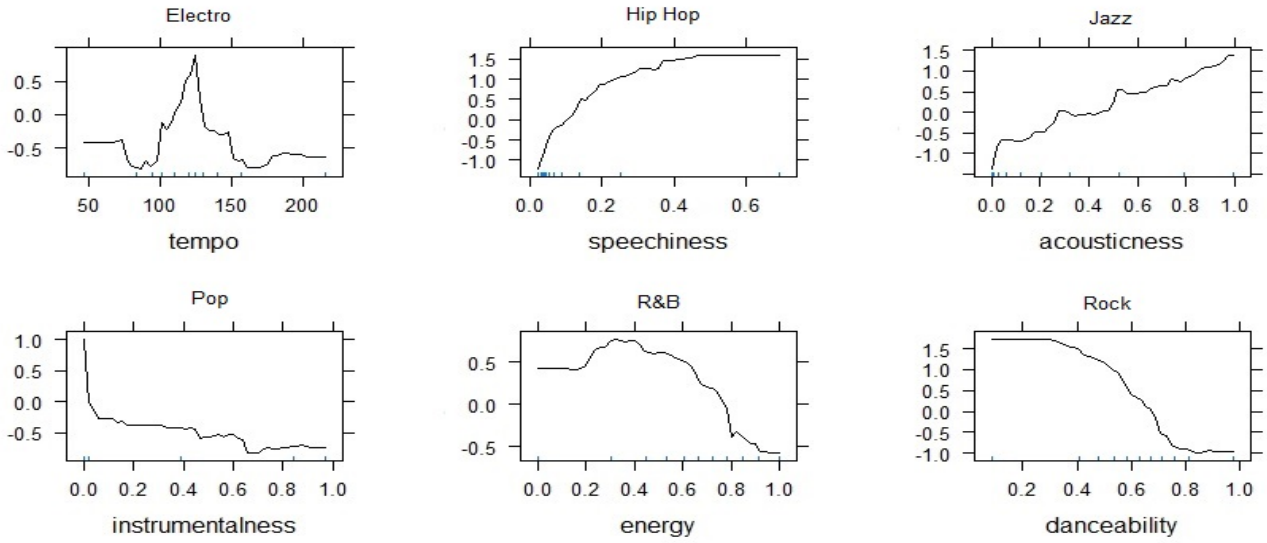


Figure 6: Partial dependence plots of the most important feature according to Gain for each class.

- Rock - danceability

The most important features for Electro, Hip hop and Jazz are perhaps the ones one would expect to see while the others are a bit less obvious. In order to understand the effect these features have on the respective classes we look at the partial dependence plots in Figure 6. These plots show the effect of a chosen feature after accounting for the average effects of all other features. On the x-axis we have the values of the features with the blue lines showing the deciles of the values of the feature in training data. This means that there is less data available where there's a large distance between the blue lines. On the y-axis we have the logits, using the softmax function (4) meaning that higher values indicate a larger probability of an observation belonging to the given class, Friedman [6].

The partial dependence plots for classes Hip Hop and Jazz look pretty much as one would expect with higher values of speechiness corresponding to a higher probability of belonging to the class Hip Hop and likewise for Jazz and acousticness. The form of the partial dependence plots for Pop and Rock also make some sense if you consider that Pop usually contains a lot of vocal content which would mean that instrumentalness is small. Similar arguments can be made for Rock and danceability where rock is generally not considered as dancing music. These four classes, especially Hip Hop, Jazz and Rock, also show overpowering importance for these features in Figure 8 of the appendix. For Electro and R&B we see a more even distribution of feature importances which is probably why we see lower ranges on the y-axis in their partial dependence plots. It is also

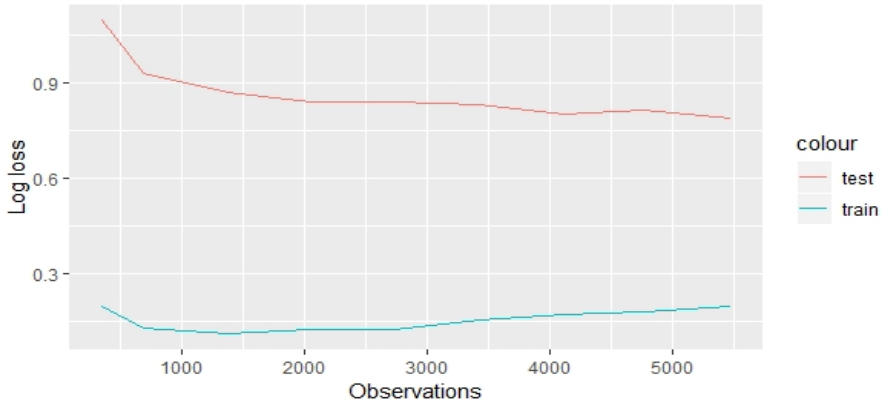


Figure 7: Learning curve plotted with cross-validated training and test log loss for 8 values between 344 and 5481 observations.

worth pointing out that, as mentioned above, large distances between blue lines in plots means less training data in those ranges, and consequently that those parts of the plots are less accurate. This includes for example the rightmost part of the Hip Hop speechiness plot or the leftmost part of the R&B energy plot.

6.5 Potential improvements

As mentioned in Section 6.2 the performance of the model is deemed to be relatively good but there is of course room for improvements. Unsurprisingly as extreme gradient boosting learns from provided training data a lot of the improvements can potentially be made here. More often than not, if one has a sufficiently good algorithm more data means better performance in supervised learning problems and considering that our training data only contains roughly 5500 observations this is almost certainly the case for us. This can be illustrated using a learning curve, such a plot can be seen in Figure 7 for our model. We have plotted the cross-validated training and test log loss using an increasing number of observations. As we can see it seems that the curve for the log loss of the testing is still on a downward trend as we reach our maximum number of observations, indicating that adding more observations could keep reducing the loss.

Other than just increasing the size of the data set there are obvious improvements to be made in the quality of the data set. Considering that more than 50% of data in the variables based on the lyrics is missing a better way of gathering these as well as a better utilization of lyrics (since we are currently only using 29 words) would probably improve the performance of the model.

When it comes to the modelling, other than a more theoretically based way of tuning the parameters of the model, it is possible that multi-label classification

may be more appropriate when classifying songs according to genres as they can contain a mix of genres as well as be considered different in terms genre by different people.

All these things will surely improve the accuracy of a model like this but to what extent is unclear. It is unlikely that creating models with close to 100% accuracy is plausible as long as the problem with ambiguity in the labeling exists, which seems like a problem inherent for these types of data sets.

7 Conclusion

The main aim of this thesis was to create a classifier for music genres, using XGboost and our Spotify data set, with satisfactory accuracy. Reaching an overall model accuracy of 73.43% is under the inherent difficulties of classifying songs according to genres and the somewhat lacking data set considered a satisfactory result even if it would not be sufficient in real world applications. Upon writing the thesis it became clear that XGboost is a very powerful method for problems like this even though it seems there is much left to be explored when it comes to tuning the parameters of an XGboost model.

The thesis also demonstrates the, perhaps expected, differences in difficulty in classification of different genres, with Rock, Jazz, Hip Hop and Electro being easier than Pop and R&B. We have also shown that several classes are correlated with one specific predictor, with some correlations being more predictable than others, such as Hip Hop's strong dependence on speechiness while R&B seems to have weaker dependencies with several predictors.

8 Appendix

Figure 8: Classwise feature importance for all six genres. The ten most important features according to the measures Gain, Cover and Frequency.

9 References

References

[1] Allison, P. D. (2002). *Missing data.* Thousand Oaks, CA: SAGE Publications, Inc. doi: 10.4135/9781412985079

[2] Bahuleyan, H. (2018). Music genre classification using machine learning techniques. Available at: https://arxiv.org/abs/1804.01149.

[3] Bergstra, J. & Bergio, Y. (2012). Random search for Hyper-Parameter optimization. *Journal of Machine Learning Research* 13 pp. 281-305.

[4] Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984). *Classification and regression trees*, Wadsworth, New York.

[5] Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. Available at : https://arxiv.org/abs/1603.02754.

[6] Friedman, Jerome H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist.* 29, no. 5 pp. 1189-1232.

[7] Hastie, T., Tibshirani R. & Friedman, J. (2017). *The Elements of Statistical Learning.* Second edition, Springer Series in Statistics Springer New York Inc., New York, NY, USA.

[8] Hossin, M. & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining Knowledge Management Process.* Available at: https://www.academia.edu/37173439/

[9] Hörnqvist, A. (2019). Living with trees - Predicting Swedish apartment prices with eXtreme Gradient Boosting. Available at: https://www.math.su.se/publikationer/uppsatsarkiv/tidigare-examensarbeten-i-matematisk-statistik/kandidat/kandidatarbeten-i-matematisk-statistik-2019-1.422151.

[10] Lundberg, S. M., Erion, G. G. & Lee, S. I. (2018). Consistent Individualized feature attribution for tree ensembles. Available at: https://arxiv.org/abs/1802.03888.

[11] McKay, C. & Fujinaga, I. (2006). Musical genre classification: Is it worth pursuing and how can it be improved? Available at: https://www.semanticscholar.org/paper/Musical-genre-classification%3A-Is-it-worth-pursuing-McKay-Fujinaga/298bee3d5bc640f335820e23bc91bb53919d798e

[12] Spotify Audio features. Available at: https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/.

[13] Spotify Web API. Available at: https://developer.spotify.com/documentation/web-api/.

[14] Thakur, A. (2016). Approaching (almost) any Machine Learning problem. Available at: https://www.linkedin.com/pulse/approaching-almost-any-machine-learning-problem-abhishek-thakur

[15] XGboost documentation (2016). Available at: https://xgboost.readthedocs.io/en/latest/.

[16] Zhang, O. (2015). Tips for data science competitions. Available at: https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions.