Music Genre Classification

Group 12

Arjan Jawahier (s2762161)
Roeland Lindhout (s2954524)
B.K. Musuadhi Rajan (s4065476)
Hari Vidharth (s4031180)

December 28, 2020

Abstract

Music genre classification is a common supervised learning task, yet due to its natural complexity, analyzing and finding patterns in the limited available data is challenging. To overcome this hurdle, Defferrard et al. have introduced the Free Music Archive dataset [Def+16] by providing 106574 tracks from 14854 albums by 16341 artists of 161 genres. Features have been extracted from these songs with the use of the LibROSA package [Lib19]. These features are used in the classification of the genre of individual songs. We try out different classification algorithms, namely linear regression, ridge regression, random forests, k-nearest neighbours and support vector machines. From our results, we conclude that the best classification was achieved using SVM with a test accuracy of 60%.

1 Introduction

The genre of a song plays a considerable role in which the audience will listen to it. People have a liking towards a specific kind or type of genre and hence often search for a specific kind of music online. Intuitively, a person who likes heavy-metal is unlikely to want to listen to children's songs and vice-versa. The probability of this person searching for music with queries such as "Best heavy-metal songs" is not low. The importance of classifying songs into different genres can be seen.

Classifying songs into genres can be very straightforward for humans most of the time. Pop-songs usually have pop-elements. Rock-songs have their rock-elements, etc. These songs are then classified into genres by people. This can easily be done since genres are defined and described by humans themselves. It is no natural phenomenon one needs to precisely model [Sam89]. However, this comes at the cost of time and money. The person who classifies each song has to listen to them first. Then, the person has to think about social conventions and current trends, as genres tend to change over time. New genres can be invented, and old genres can be discontinued [J S12]. This process invites the use of Machine Learning (ML) techniques, as it is a classification task.

Using classifiers (in the ML sense) seems like a good way to cut down on costs needed for genre-tagging. A good classifier would hopefully generate the same genres for songs as humans would. By testing the classifier on unseen data, we can decide how well it does compared to humans.

The dataset we used is a dump of the Free Music Archive (FMA), an interactive library of high-quality, legal audio downloads. The original goal of the creators of the FMA was to create an open-source and easily accessible large dataset, suitable for evaluating several tasks in MIR, a field concerned with browsing, searching, and organizing large music collections [Def+16]. It was made for the community whose growing interest in these kinds of tasks were hindered by the lack of availability of a free open and large data. FMA aims to overcome this hurdle by providing 917 GiB and 343 days worth of Creative Commons-licensed audio from 106,574 tracks from 16,341 artists and 14,854 albums, arranged in a hierarchical taxonomy of 161 genres. It provides full-length and high-quality audio, pre-computed features, together with track-level and user-level metadata, tags, and free-form text such as biographies [Def+16].

2 Methods

Algorithm 1 shows pseudo-code for the highest level view of the main pipeline. Each of these steps are described in this section.

Algorithm 1: A high level view of the main pipeline.

Result: Test accuracy
data $\leftarrow ReadDataset(data)$
data $\leftarrow CleanData(data)$
data $\leftarrow ConvertMultiLabelToSingleLabel(data)$
data $\leftarrow RemoveInfrequentClasses(data)$
data $\leftarrow SelectFeatures(data)$
train_data, test_data $\leftarrow SplitTrainTest(data)$
classifier $\leftarrow DefineClassifier$
classifier $\leftarrow TrainClassifier(classifier)$
test_accuracy $\leftarrow EvaluateClassifier(classifier)$

2.1 Reading the dataset

The dataset used for the music genre recognition is the FMA dataset [Def+16]. This dataset consists of multiple downloadable Comma-Separated-Values (CSV) files. Two of these CSV-files were of interest in this case.

The file *features.csv* contains the features of 106574 songs. These features were extracted from each song with a Python package built specifically for music and sound analysis named LibROSA [Lib19]. Each song is represented as a feature vector with 518 feature values.

The file *tracks.csv* contains the genre codes of all songs in the dataset, among other things. Since we only need the genre codes as class labels, only that specific column is read.

2.2 Cleaning the data

The dataset is not entirely clean. Some of the songs in *tracks.csv* have shifted column entries. As a consequence, the values in the genres column are not valid genre codes. Furthermore, some of the songs do not even have a listed genre. In both cases, the song was removed from the data.

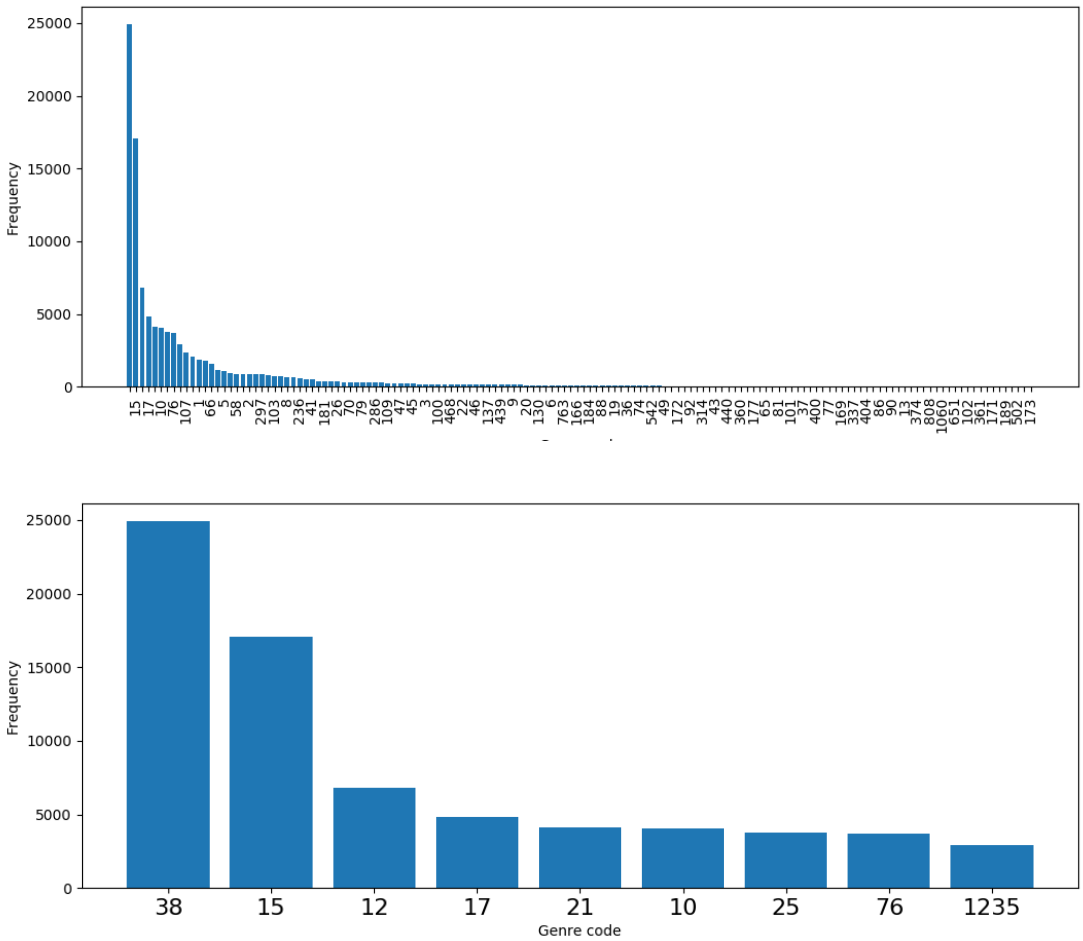

Figure 1: **Top**: Vertical bar charts of frequencies of genre codes. As can be seen, the distribution of genres is highly non-uniform, which creates a bias in the decision function for more frequently occurring genres. The genre codes were adopted from [Def+16]. Note: only the genre code for each second genre is labeled. This was done to improve clarity. **Bottom**: Vertical bar charts of frequencies of genre codes after removing infrequent classes. Each genre now has at least 2500 instances. This results in nine classes still being present in the dataset.

2.2.1 Converting multi-label classes to single-label

The entries in the genres column of *tracks.csv* are lists of integers which indicates that one song can be of a variable number of genres. We converted songs with more than 1 genre to only 1 genre. In this case, random selection would not at all prove beneficial for training the classifiers, because the random selection of genre would lead to an unbalanced dataset with a different number of examples for different genres which would keep changing for every run. To fix this, we implemented an algorithm that picks one genre from the given multiple genres: For each genre, the frequency is measured. Among the different genres presented in a row that has more that one genre, the genre is picked that is the most frequent among these genres. This has proven useful when experimenting with the threshold values for genre selection in the following sections by providing more training and validation examples. This led to the creation of a more robust classifier.

2.3 Removing infrequent classes

Figure 1 (Top) shows a bar chart of the genre codes after cleaning the dataset up to this point. The class distribution is highly irregular, which can pose problems like mapping an infrequent class to a frequent class in the decision functions later. Many classes had few training examples which proved to be a struggle for the learning algorithms. To fix this issue, we discarded all the genres with fewer instances than a certain threshold of 2500. Consequently, we ended up with the bar chart in Figure 1 (Bottom). During this process, the number of classes diminished from 161 to nine. The genres that are left are (genre code, genre name): (10, Pop), (12, Rock), (15, Electronic), (17, Folk), (21, Hip-Hop), (25, Post-Rock), (38, Experimental), (76, Experimental Pop) and (1235, Instrumental). Our classifiers will thus try to differentiate between these nine classes.

2.4 Feature selection

Dimension reduction can be an essential step in the machine learning pipeline. Too many dimensions (and thus many learn-able parameters) can cause the learner to overfit. Moreover, using all features can slow down the learning process significantly. The shape of the data after removing the infrequent classes is (72104, 518). Some learning algorithms will take hours or even days with this much data.

Principal Component Analysis (PCA) was applied to the dataset. This method found out the features that have the highest variance and which features are thus most important when the goal is to keep as much information as possible in the case of dimension reduction. However, the features that display the most variance are not necessarily the most important features for our goal of music genre classification. It could certainly be the case that some of the features that have little variance are more important for the classification of music genres than features that have large variances. This method was thus replaced by another.

As mentioned before, the feature vectors are concatenated features extracted with the use of the LibROSA API [Lib19]. Whereas the documentation of LibROSA mentions 14 different feature vectors, the FMA dataset consists of only 11 of these feature vectors (concatenated). For each combination of extracted feature vectors of length 1, 2 and 3, linear regression classification has been performed on the data. Then, the combination of feature vectors that yielded the highest mean cross-validation score was chosen as "the best feature combination". The best combination of features that was obtained is:

- **chroma_cens**: A chromagram variant where the chroma energy is normalized [Lib19].
- **mfcc**: Mel-frequency cepstral coefficients [Lib19].
- **spectral_contrast**: Contrast of spectral energy [Lib19].

Using only these three concatenated feature vectors instead of all features diminished the number of dimensions of the data from 518 to 273, which means 245 features were discarded.

2.5 Applied learning algorithms

The decision was made to try multiple learning algorithms for the classification of musical genres, and report on the best performing one. This is by no means an exhaustive search for the best algorithm. The approach used here can be seen as an exploratory baseline search, before more complicated algorithms would be used, such as Deep Learning. The learning algorithms that we used are Ridge/Linear regression, Random Forests, Support Vector Machines, and K-Nearest-Neighbors.

2.5.1 Linear/Ridge regression

Linear regression tries to find the best fitting hyperplanes that separate decision regions. It does this by tuning the parameters of the hyperplanes such that the total training error is minimized [Ped+11]:

$$\min_w ||Xw - y||_2^2. \tag{1}$$

Ridge regression is a modified version of linear regression. The only difference is a penalizing factor of α. This factor penalizes high slopes (i.e. high parameter values), which causes the slopes in ridge regression to not be as steep as their linear regression counterparts. The linear regression expression becomes [Ped+11]:

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2. \tag{2}$$

The higher α is set to, the lower the slopes. Ridge regression essentially trades variance for bias. When α approaches infinity, the decision hyperplanes will have shallow slopes (approaching 0), but different biases. When α is equal to 0, there is no penalizing going on, so this case is equivalent to linear regression. It is important to find the correct value for α as values that are too low can cause overfitting, whereas values that are too high can cause underfitting [Bri16]. Figure 2 shows the effect of choosing different values for α in the 2-dimensional case.

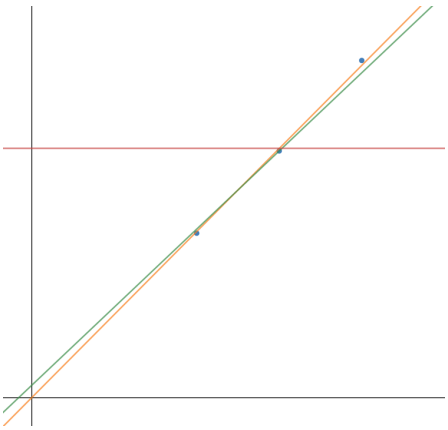

Figure 2: 2-Dimensional case of Ridge regression. The 3 data points (blue) can be fitted with various lines with Ridge regression. Varying the value of α has an effect on the slope and y-intercept of the fitted line. Orange line: $\alpha = 0$, equivalent to the linear regression case. Green line: α is small, the slope is less steep but bias (y-intercept) is higher than in the linear regression case. Red line: $\alpha = \infty$, the line has no slope but a lot more bias.

Four different Ridge regression classifiers have been trained, each with a different value for α. The values that were used were 1, 2, 3 and 4.

2.5.2 Random Forest

Random forests can be used to classify data by creating decision trees, which splits the data based on the most relevant features and to do this, the information gain of the different features is calculated. The feature that has the highest information gain is chosen as a new node to separate the data. An ensemble of such decision trees, which all were created using a different selection of the data, using "bagging", can be used to make decisions, using a majority vote.

An initial exploration of the random forest approach on this data showed that the Forest was overfitting. This could be seen from the fact that the training accuracy was very high, at approximately 99%, while the testing error was only at approximately 40%.

Different parameters can be altered to cope with this problem. One option is to use Minimal Cost-Complexity pruning, in which the nodes which have the highest rate of misclassification are pruned to reduce the amount of overfitting. Several degrees of pruning were tested, from 0 to 10%. It quickly became clear that the pruning led to a decrease in performance. Although the decrease in performance was smaller for the testing set than for the training set, it was concluded that this would not improve the overall system.

Other parameters to cope with the overfitting problem are the maximal depth of the trees and the minimal impurity decrease. By reducing the maximal depth, the decision criteria of the trees become less specified on the training data. Increasing the minimal impurity decrease leads to nodes that only split up the data when the decision leads to an increasing amount of impurity decrease, i.e., a higher information gain.

A combination of these parameters was tested. The maximal depth parameter was tested on a range from 14 to 28 nodes deep with a step-size of two, and the minimal impurity decrease was tested on a range from 0 to 1E-7, with a step-size of 2E-8. The resulting forests using these parameters were compared to each other using 10-fold cross-validation. The highest performance was found using a maximal depth of 28 nodes, and a minimal impurity decrease of 2E-8. Although this combination of parameters gave the highest performance, the differences between the combinations of features were not very large.

Another parameter that can alter the performance of the random forest is the impurity measure. Besides the regular information gain, the *Gini impurity* was also tested. *Gini impurity* measures error rate when a category decision for a randomly picked point is made randomly according to the class distribution. The performance of the random forest was not improved by using the *Gini Impurity*.

In conclusion, the parameters that were used for random forests are no pruning, maximal depth of 28, and minimal impurity decrease of 2E-8. The random forest was trained on the three selected features from the feature selection.

2.5.3 Support Vector Machine

Support Vector Machines (SVM or SVMs) for short are one of the more robust classifiers which are preferred for complex multi-class classification tasks. SVMs are non-probabilistic binary linear classifiers. The SVMs classify the data by constructing a hyperplane that has the largest distance to the nearest data points, the so-called support vectors. The SVM can classify the data points using these margins and constructing the hyperplane.

It is also known as a large margin classifier, the larger the margin, the lower is the generalization error. In addition to the linear classification, SVMs can efficiently perform non-linear classification using the kernel trick, by mapping the inputs to higher dimensional feature spaces and constructing the hyperplane of separation in the higher dimensions.

Experimenting with the default parameters for SVM, yielded the results of training accuracy of 70% and test accuracy of 40%. The learning was going well, but the SVM could not generalize enough. It was clear that hyperparameter tuning for the SVM was needed and also some preprocessing for the dataset.

For preprocessing steps, the dataset was normalized but did not yield good results; instead, the scaling method was used. Scikit-learn's StandardScaler [Ped+11] method was used to scale the data.

$$z = (x - \mu)/\sigma \tag{3}$$

In Equation 3, z is the standard score used for scaling, x is the data point, μ is the mean value of the feature vector, and σ is the standard deviation. Out of the different scaling methods [Ped+11] StandardScaler, MinMaxScaler, RobustScaler, the StandardScaler was found to be the best. The classes were stratified between the train and the test sets, meaning the examples for the classes where split equally among the train and test sets. The labels were encoded into an ordered numerical category using scikit learns [Ped+11] Label encoder method and the SVM takes care of one-hot encoding the labels internally.

For the hyperparameters tuning, Scikit-learn's [Ped+11] built in tool/framework known as GridSearchCV was used to test out the different hyperparameters. The choices for tuning the hyperparameters were C value, gamma value, kernel type, degree (Only in the case of Polynomial kernel). The value of C is the strength of the regularization parameter, the value of C should always be positive, and the penalty is a squared 'l2' penalty. The values of C chosen for testing were [1.0, 10.0, 100.0] out of which 10.0 was giving the best results. The value of gamma is a kernel coefficient; the values of gamma chosen for testing were [0.1, 0.01, 0.001, 'scale', 'auto']. Equations 4 and 5 are used to calculate the 'auto' and 'scale' values:

$$Auto = 1/N \tag{4}$$

$$Scale = 1/(N * \sigma^2). \tag{5}$$

In both equations, N stands for the number of features and σ^2 stands for the variance of those features.

The kernel type parameter represents the type of kernel; the above-mentioned baseline was taken with the linear kernel, hence for this test, the choice of the kernel was 'RBF' (Gaussian) and polynomial which was tested with different degrees.

In conclusion, from the above testing, the best results were obtained with hyperparameters 'RBF' kernel, C value of 10.0, and gamma value of 'scale'.

2.5.4 K-Nearest-Neighbours

K-Nearest Neighbours (kNN or kNNs) is a non-parametric distance-based learning method for solving classification problems. For each test input pattern, the algorithm finds the k nearest neighbours, based on the chosen distance metric. If $k = 1$ the new data point is assigned to the closest neighbour. In $k > 1$ cases there is a majority vote among the k nearest neighbours and the new data point belongs to the class with the maximum vote. Here the distance metric and the value of k are the hyperparameter needed to be tuned to get the best results.

As mentioned above in SVM classifier, the experiments were performed on the cleaned dataset. Normalization of the values did not produce good results; hence Standard-scaling was done as mentioned in the formula above. The distance metric was chosen as 'minkowski' as this was found to be the best compared to Euclidean and city block. As for the value of k we chose [1, 3, 5, 7] and using the elbow method observed no improvement beyond $k > 7$ and hence these were the four values of k that was considered for this experiment.

3 Results

To see which of the algorithms was best able to classify the music into the correct genres, their results are shown in Table 1. Other than the choice of algorithm, the situation for the training and testing of all different approaches remained the same.

In Table 1, the proportion of correctly labelled genres on the test set of the different learning algorithms are shown. Initially it can be seen that the performances do not differ much between the different algorithms, as the accuracies are all around 0.5, apart from the SVM, which had a test accuracy of 0.60.

In the case of SVM, out of all the hyperparameters mentioned above, the best results were obtained with 'RBF' kernel, C value of 10.0, and gamma value of 'scale'. The SVM scored a training accuracy of 0.96 and a

Learning algorithm	Test accuracy (proportion)
Linear Regression	0.5134
Ridge Regression ($\alpha = 1$)	0.5140
Ridge Regression ($\alpha = 2$)	0.5141
Ridge Regression ($\alpha = 3$)	0.5143
Ridge Regression ($\alpha = 4$)	0.5141
Random Forest	0.5085
K-nearest-neighbors ($k = 1$)	0.4539
K-nearest-neighbors ($k = 3$)	0.4674
K-nearest-neighbors ($k = 5$)	0.4666
K-nearest-neighbors ($k = 7$)	0.4679
SVM	**0.6029**

Table 1: The test results (accuracy on unseen data) for each learning algorithm, some with specified parameters. The best result is highlighted in bold font.

test accuracy of 0.60. The chosen parameters provided the best possible fit for the data with better generalization.

The best SVM classifier produced the following confusion matrix:

$$\begin{pmatrix} Actual \backslash Predicted & 10 & 12 & 15 & 17 & 21 & 25 & 38 & 76 & 1235 \\ Pop(10) & \mathbf{289} & 75 & 63 & 90 & 12 & 24 & 47 & 43 & 11 \\ Rock(12) & 113 & \mathbf{870} & 70 & 69 & 11 & 193 & 144 & 78 & 26 \\ Electronic(15) & 137 & 58 & \mathbf{2101} & 56 & 202 & 34 & 831 & 159 & 158 \\ Folk(17) & 82 & 61 & 34 & \mathbf{568} & 5 & 4 & 88 & 36 & 27 \\ Hip-Hop(21) & 25 & 12 & 126 & 5 & \mathbf{495} & 9 & 52 & 14 & 9 \\ Punk(25) & 12 & 80 & 12 & 5 & 2 & \mathbf{395} & 33 & 13 & 8 \\ Experimental(38) & 94 & 170 & 931 & 135 & 86 & 82 & \mathbf{3662} & 255 & 152 \\ Experimental-Pop(76) & 35 & 26 & 35 & 21 & 9 & 4 & 73 & \mathbf{133} & 7 \\ Instrumental(1235) & 18 & 15 & 47 & 10 & 2 & 2 & 53 & 6 & \mathbf{182} \end{pmatrix}$$

From this matrix, the per-class accuracies can also be calculated by dividing the boldfaced number in each row by the sum of that row. This leads to the class accuracies as laid out in Table 2.

Genre	Class accuracy
Pop	0.441
Rock	0.553
Electronic	0.562
Folk	0.628
Hip-Hop	0.663
Punk	0.705
Experimental	0.658
Experimental-Pop	0.388
Instrumental	0.543

Table 2: Per class accuracies. It is clear that for some classes, the SVM is performing quite poorly, whereas for others, it is performing well above average.

4 Discussion

In this paper, various approaches were used to classify the songs from the FMA dataset [Def+16]. This dataset consists of over 100,000 songs, of various genres. The goal of this research was to find an optimal learning algorithm from the following algorithms: Linear/Ridge Regression, Random Forest, Support-Vector Machine and k-Nearest-Neighbours.

Before the learning algorithms could be applied, the dataset was inspected. By inspecting the dataset, it became clear that it was a highly unbalanced dataset. Some genres have copious amounts of songs, whilst other genres are very infrequently represented. To deal with this, a threshold for a minimal number of songs per genre was set. Genres with fewer songs were removed from the dataset. The genres that were removed were often sub-genres, which are also contained in other genres. This helped in reducing the complexity of the task, as the number of genres that the songs could be classified into was reduced but at the cost of losing a lot of minor genres during this process.

Another design decision was to simplify the task from multi-class labels to single-class to reduce the difficulty of the classification task. Songs from a particular sub-genre, such as Punk, which is a sub-genre of Rock, were classified as their top-level genre. By ensuring that the sub-genres were classified as their top-level genre, the classification of the songs was simplified, since it increased the number of songs per genre while reducing the number of possible genres the songs could be classified as.

Feature selection plays a vital role in the performance of a classifier in case of a large number of dimensions. From the songs in the dataset, a number of features were extracted. These features could all play a role in classifying the songs into the correct genres. However, more features could lead to overfitting, as the information about the songs becomes too detailed. Another problem with using all features, is that the learning algorithm takes a long time. To cope with this problem, combinations of the different features were applied to the linear regression. The combination of chroma_cens, mfcc and spectral_contrast proved to be the best combination of features. It reduced the dimensions of the data from 518 to 273 dimensions. This is significant change in dimensions, but 273 dimensions still seems like a high number. This begs the question: Is it not possible to reduce the number of dimensions even more without significant loss of information? Perhaps the answer lies in using different features altogether.

Finally, the different algorithms were trained using the simplified dataset and the selected features. Surprisingly, the performances of the linear regression, the random forest and k-Nearest-Neighbours were very similar. The SVM approach performed the best on unseen data. The main advantage of SVM over the other classifiers is that the SVM methodology can be paired with the Kernel trick. By exploring and tuning the kernels, the dataset can be transformed into N higher dimensional feature spaces as needed where the linear classifier can classify the data created by non-linear circumstances. Even though this is the case, 60% accuracy is not that great. Furthermore, the training accuracy was a lot larger, which indicates a large amount of overfitting. This situation could have been improved by using more regularization techniques, such as adding noise to the data. As a final note, it is clear that this task is complicated, even with all the simplifications that were introduced.

References

[Bri16] Brilliant.org. *Ridge Regression*. [Online; accessed 2-February-2020]. 2016. URL: `https://brilliant.org/wiki/ridge-regression/`.

[Def+16] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson. "FMA: A Dataset For Music Analysis". In: (2016). eprint: `1612.01840`. URL: `https://arxiv.org/abs/1612.01840`.

[J S12] J. Samson. *Genre*. [Online; accessed 2-February-2020]. 2012. URL: `https://doi.org/10.1093/gmo/9781561592630.article.40599`.

[Lib19] LibROSA contributors. *LibROSA Documentation*. [Online; accessed 25-January-2020]. 2019. URL: `https://librosa.github.io/librosa/`.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pages 2825–2830.

[Sam89] J. Samson. "Chopin and genre". In: *Music Analysis* 8.3 (1989), pages 213–231.