COMS3007 Machine Learning Assignment

May 21, 2019

Tristan Nagan - 1484720 Marc Marsden - 1437889

Lehyendran Govender -1106458 Theo Scott van Zyl - 1240639

1 Introduction

This project implemented machine learning methods to differentiate music between different composers. This was achieved by inspection of MIDI metadata and audio features (predominantly consisting of spectral analysis). The first two models were different versions of **Naïve Bayes** and the third was **Logistic Regression**, which gives the probability that a certain composition is composed by a certain composer. After passing our data through 3 algorithms, we ran multiple tests using different combinations of composers and features in order to find the combinations that gave us the highest accuracies.
Finally we analysed the performance of our algorithms and then added some recommendations for working with such data.

2 Dataset Description

This project was based on a public set of classical compositions for piano. The dataset was sourced from http://www.piano-midi.de/. MIDI files were taken as the raw data and **jAudio** (http://jaudio.sourceforge.net/) was used to extract features from the MIDI. There are 127 MIDI files (datapoints) in the dataset. The size of the dataset was increased by splitting the MIDI files into 16 second samples before extracting 13 audio features per file. This gave us a total of 2514 samples.

2.1 Attributes

The chosen target variable was composer name.

Target Classes
Beethoven
Chopin
Mozart
Schubert

The extracted audio attributes/features using **jAudio**:

Features	Description
MFCC	The Mel-frequency Cepstrum (MFC) is a representation of the short-term power spectrum of a sound, the Mel-frequency Cepstral Coefficients (MFCCs) are coefficients that collectively make up an MFC.
Spectral Flux	A measure of how quickly the power spectrum of a signal is changing
Compactness	A measure of the noisiness of a signal. Found by comparing the components of a window's magnitude spectrum with the magnitude spectrum of its neighbouring windows.
Spectral Variability	The standard deviation of the magnitude spectrum. This is a measure of the variance of a signal's magnitude spectrum
Root Mean Square	(RMS) is a measure of the power of a signal.
Zero Crossings	The number of times the waveform changed sign. An indication of frequency as well as noisiness.
Strongest Frequency Via Zero Crossings	The strongest frequency component of a signal, in Hz, found via the number of zero-crossings.
Strongest Frequency Via Spectral Centroid	The strongest frequency component of a signal, in Hz, found via the spectral centroid.
Strongest Frequency Via FFT Maximum	The strongest frequency component of a signal, in Hz, found via finding the FFT bin with the highest power.
LPC	Linear Prediction Coeffecients calculated using autocorrelation and Levinson-Durbin recursion.
Method of Moments	Statistical Method of Moments of the Magnitude Spectrum.
Relative Difference Function	Log of the derivative of RMS. Used for onset detection.
Peak Based Spectral Smoothness	Peak Based Spectral Smoothness is calculated from partials, not frequency bins.

The extracted audio attributes/features using **Mido**:

Features	Description
Key Signature	In musical notation, key signature refers to the arrangment of signs such as sharps or flats to indicate its corresponding musical notes
Time Signature	Tells us how the music is supposed to be counted

Features	Description
Mean Tempo	An average of the tempo of a composition

The last 3 attributes were only used used in the **Discrete Naïve Bayes** algorithm. Mido is a Python library for for working with MIDI Objects (https://mido.readthedocs.io/en/latest/). Please see Appendix A for an example of a data point for the main methods (**Gaussian Naïve Bayes** and **Logistic Regression**) is:
An example of a data point for the **Discrete Naïve Bayes** algorithm (after feature extraction) is:

```
['Schubert', 'Ab', '3,4', 1]
```

2.2 Data Structuring and Normalization

The data has been limited to only include the works of **Chopin**, **Mozart**, **Schubert**, and **Beethoven**. This choice was made to narrow the problem space. The data was preprocessed by passing it through **jAudio** to extract the desired features. The size of the dataset was increased by splitting the MIDI files into 16 second samples before extracting the audio features with **jAudio**. The split MIDI files were not used for the **Discrete Naïve Bayes** method since this would have lead to repeated values in the training data rather than an expanded data set. For normalisation, we passed our data through **sklearn's** StandardScaler() method. This allowed us to standardise/normalise the features by removing the mean, and then scaling to unit variance. If we did not do this, the features could have behaved very badly(i.e. a gaussian with mean = 0 and variance = 1)

2.3 Splitting the data

For the **Gaussian Naïve Bayes** and **Logistic Regression** methods, which operated on the extracted audio features, the data was split using the `train_test_split` method from the **sklearn** library. This method splits the data randomly into training and testing data according to a given ratio. We used 66.6% of the data for training and 33.3% for testing.
For the **Discrete Naïve Bayes** method, which operated on the raw MIDI files, two different strategies for splitting the data were implemented. The first strategy was to split the data into 60% training data and 40% test data by picking randomly from the available datapoints. The second strategy was to split the data in the same 60% / 40% ratio but enforced equal representation for all composers in the training data. A comparison of these two strategies is given in the **Discrete Naïve Bayes** section below.

3 Algorithms

3.1 Gaussian Naïve Bayes

This algorithm used the audio features we extracted with **jAudio**.

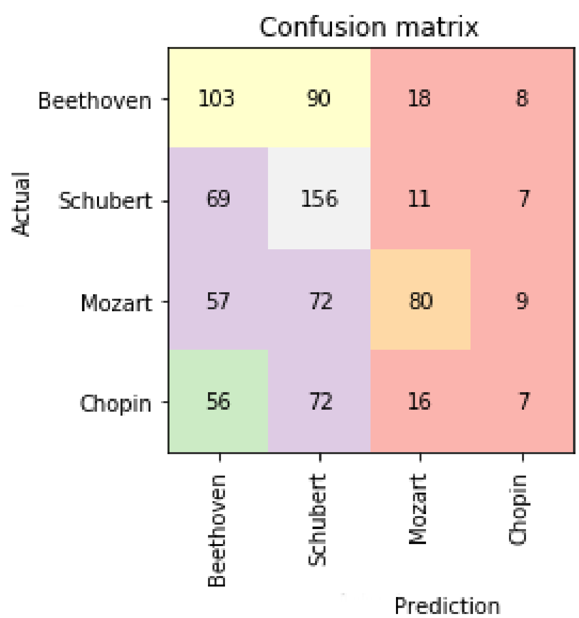
3.1.1 Implementation Details

The implementation of this algorithm assumed a normal distribution for each feature in the data. The variances and means of these distributions were learned from the training data and then used to calculate the likelihoods for the test data. When performing a classification with this algorithm, the probability of generating a given feature value within a given class needs to be calculated. Since the probability of generating any given feature value in continuous data is zero, the algorithm instead looks at the probability of generating a value within $10^{-9}\sigma^2$ of the given feature value. This interval seems sufficiently tight to reject false positives but wide enough to mitigate the problem of finding a probability of 0 for all values. The method was tweaked to ensure that the random selection gave us an equal ratio of data for the composers as a random split could be bias towards one composer.

3.1.2 Error On Test Set

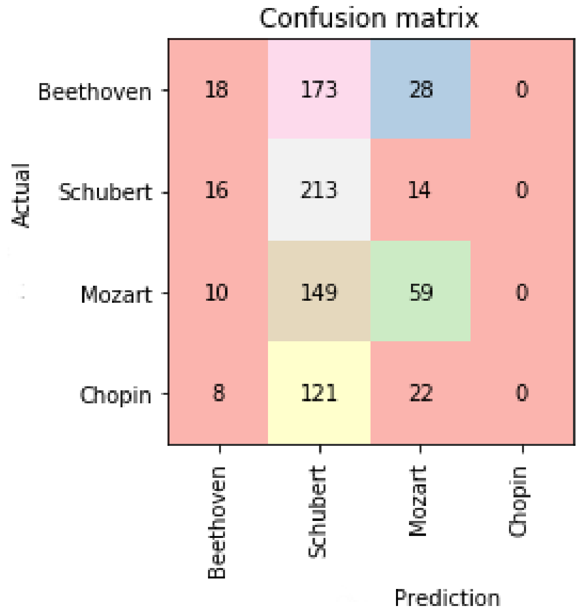
The data was tested against the individual features to see which were the best. Below are the 7 features that gave the best performances

MFCC



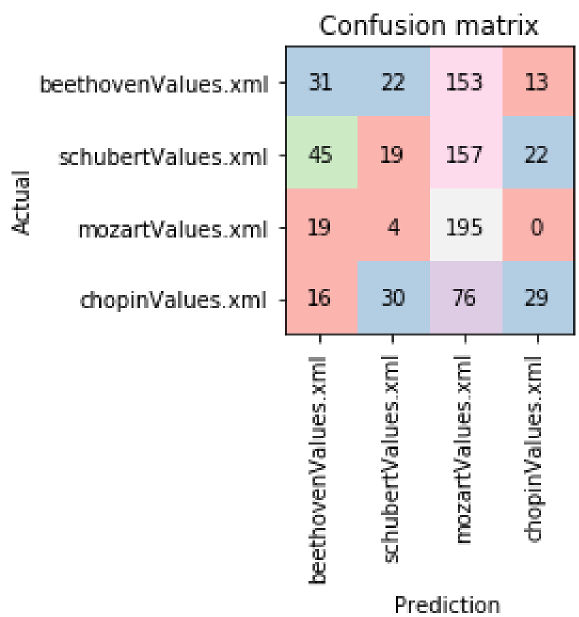
Accuracy = 41.63658243080626

Compactness



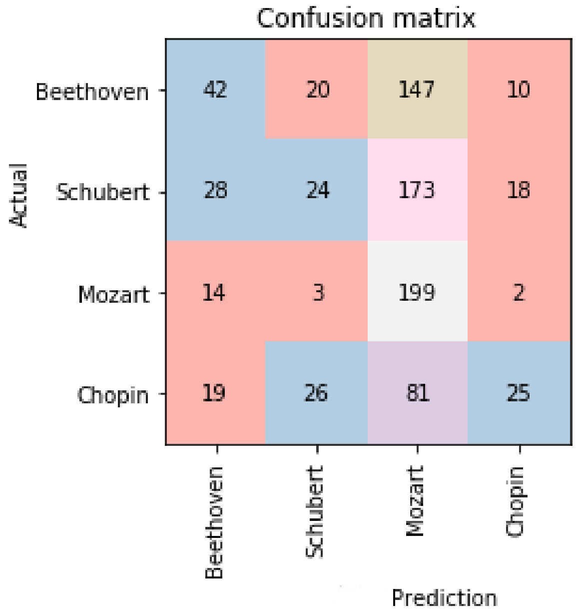
Accuracy = 34.89771359807461

Spectral Flux



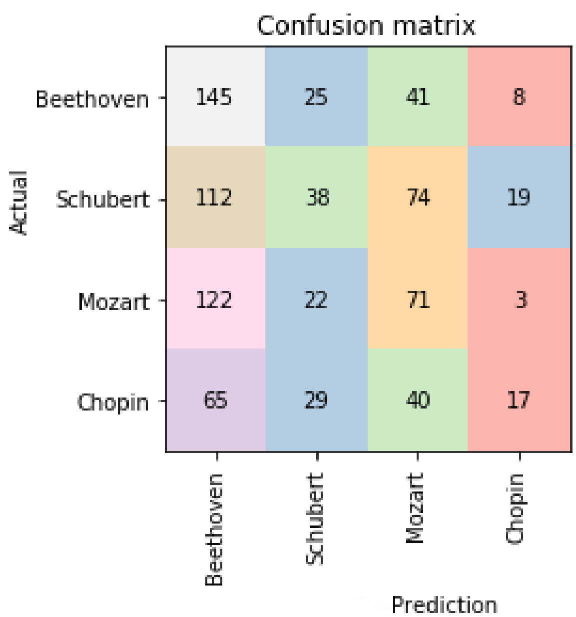
Accuracy = 34.89771359807461

Peak Speactral Smoothness



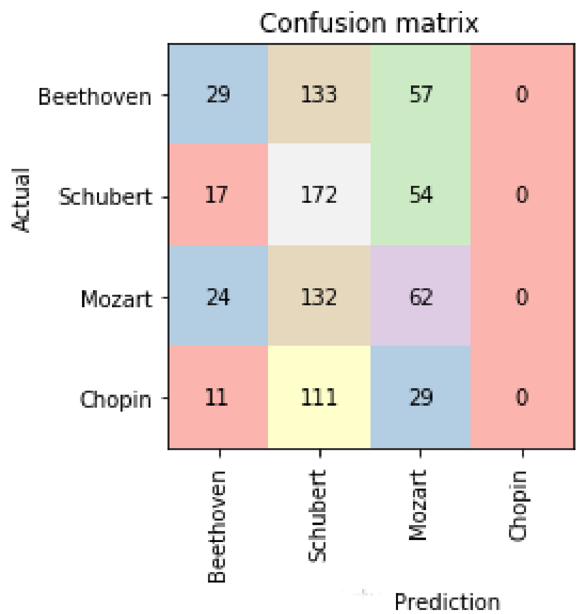
Accuracy = 34.05535499398315

Method of Moments



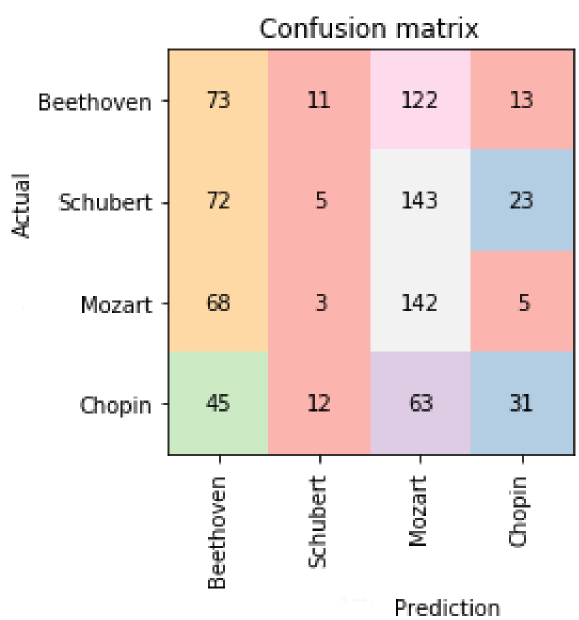
Accuracy = 32.61131167268351

Strongest Frequency Via FFT Maximum



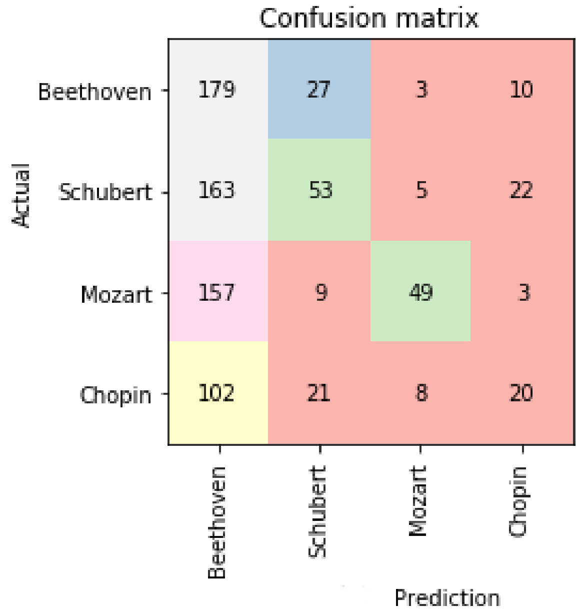
Accuracy = 31.64861612515042

Root Mean Squared



Accuracy = 30.20457280385078

All Features



Accuracy = 36.22141997593261

From the results it is easy to see that MFCC is a better feature when it is the only one used versus using all the features. This could be due to the fact that the data set was not big enough or that some of the features were not very good choices for distinguishing between different composers.

3.2 Discrete Naïve Bayes

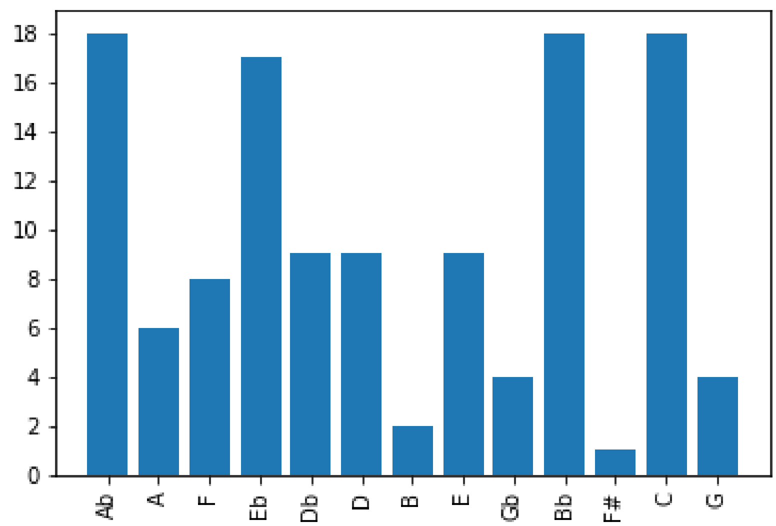
This algorithm used the metadata and message data directly from the MIDI files to predict the composer of a given piece. The reason for this implementation was to try and perform the classification with discrete data straight from the MIDI files in order to avoid inaccuracies and complications introduced in the processing of the continuous; multi-dimensional features we extracted using **jAudio**. Although it is not a particularly interesting way to view the data (since the composer of a piece is usually specified in the metadata of a MIDI file) it may provide an interesting contrast to the other method in terms of results.

3.2.1 Implementation Details

A straightforward implementation of **Naïve Bayes** with Laplace smoothing. The features used in this algorithm were extracted directly from the MIDI files and were chosen for simplicity's sake. Due to inconsistent labelling of composer names in the MIDI files, some datapoints got lost in the data preparation process and thus were not used for this algorithm. Since this algorithm used a different set of features from the other two, a brief description of these features is given below:

Key Signature

The first key signature given in the MIDI file. Subsequent key signature changes were ignored for simplicity. Key signatures are represented as strings such as `C` or `Ab`. Due to inconsistencies in the representation of key MIDI metadata, some key signatures were represented twice in two different ways (for example **F#** and **Gb** which are in fact the same key).



Occurences of different key signatures in the data

Time Signature

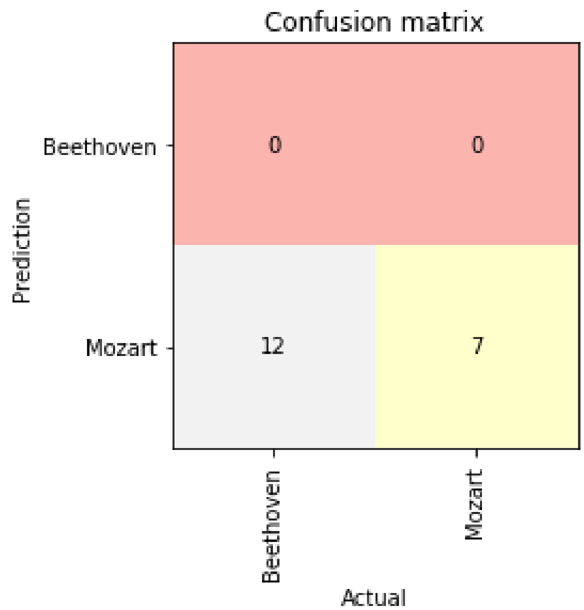
The first time signature given in the MIDI file. Subsequent time signature changes were ignored since they are fairly uncommon in the dataset, and would simply complicate the problem. Time signatures are represented as strings such as `3,4` or `5,4`.

Mean Tempo

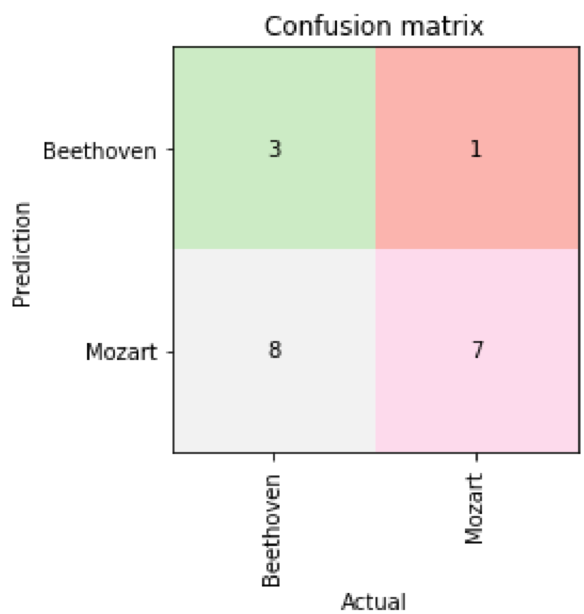
An average of the tempo throughout the whole piece. This is measured in ticks. The mean tempo was then discretized by finding the mean μ and variance σ^2 of the mean tempos across all data points and then turning them into discrete values according to the rule:
$T_{class} = 0$ if $T_{value} < \mu - \sigma^2$, Low tempo.
$T_{class} = 1$ if $\mu - \sigma^2 <= T_{value} <= \mu + \sigma^2$, Mid tempo.
$T_{class} = 2$ if $T_{value} < \mu + \sigma^2$, High tempo.
High mean tempos were absent in the dataset.

3.2.2 Error On Test Set

When the data was split randomly into training and testing data, one of the composers tended to be over-represented in the training data, leading to the model favouring that composer in the prediction. When the training data was chosen in a favourable way, the results were fairly good for the two composers case.

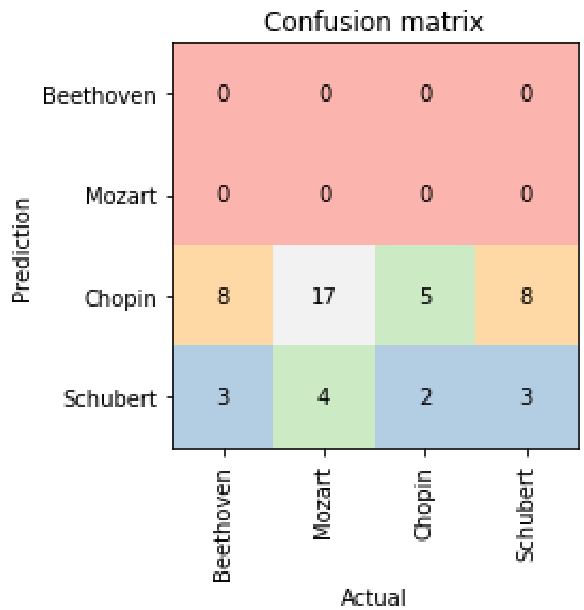


Discrete Naïve Bayes on a random test set (two composers). First result.



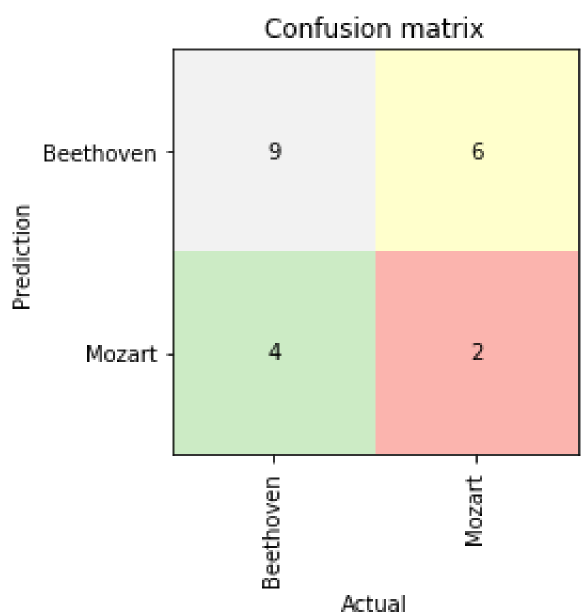
Discrete Naïve Bayes on a random test set (two composers). Second result.

The problem got worse when all four composers were present in the data. Chopin and Schubert are hugely overrepresented, so the random process tended to favour them more.

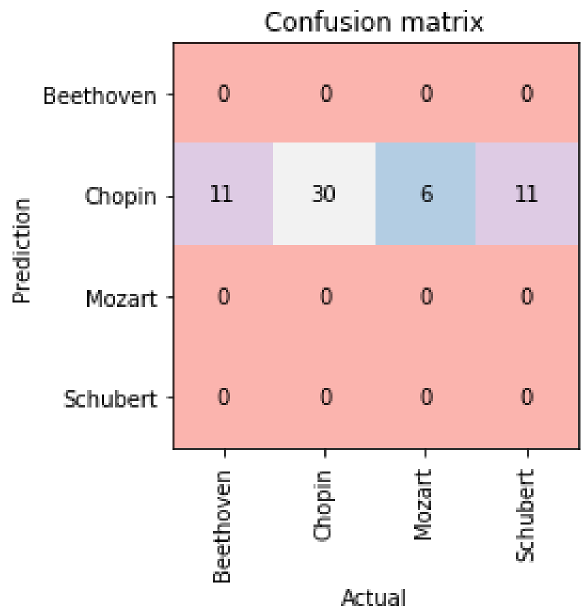


Discrete Naïve Bayes on a random test set (four composers).

When the number of datapoints for each composer in the training data was standardized, it improved the results of this algorithm slightly for the two composers case but didn't improve the results for the four composers case.



Discrete Naïve Bayes on an equalized test set (two composers).



Discrete Naïve Bayes on an equalized test set (four composers).

The lack of improvement in the four composers case serves to demonstrate that either the dataset is too small or these features are not sufficient to characterise a composer's style and thus are a poor choice compared to the extracted audio features used in the other algorithms.

3.3 Logistic Regression

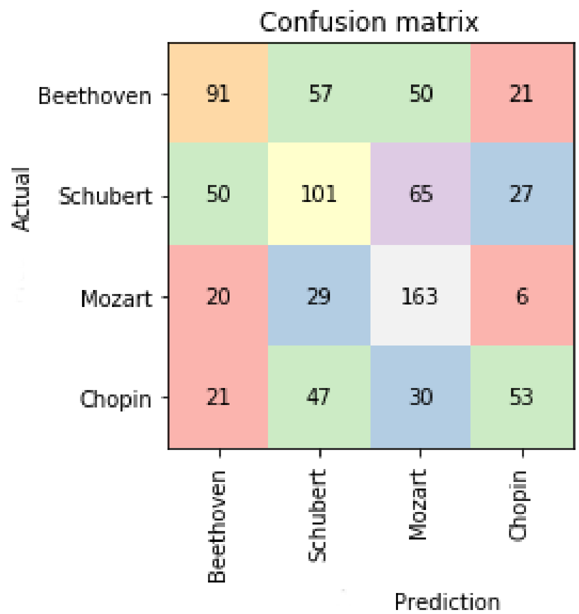
This algorithm used the same data as the **Gaussian Naïve Bayes** algorithm. Logistic regression gave us a more discriminative method in comparison to **Naïve Bayes** which is more generative. Thus, this led to a more continuous measure that provides us with a probability which represents the likelihood that a piano composition belongs to a certain composer.

3.3.1 Implementation Details

We implemented multiclass classification using the "One vs Rest(OVR)" method. Since we have 4 target classes, namely; **Beethoven**, **Chopin**, **Mozart** and **Schubert**, we first treated **Beethoven** as one class and **Chopin**, **Mozart** and **Schubert** as the other class and then ran our logistic regression model. We repeated this process for each composer and ended up with 4 different, independent logistic regressions.
We then had 4 classifiers to use for prediction where each one gave us a probability of its associated class. The most probable class is then the one that yields the highest probability.

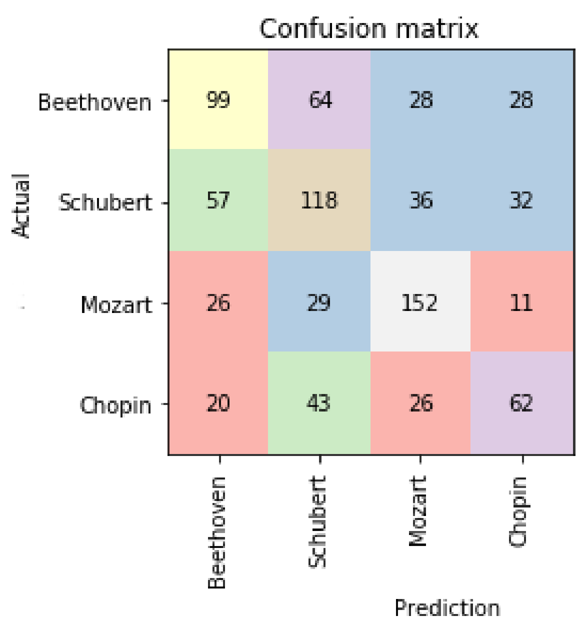
3.3.2 Hyperparameters

The learning rate, α, that we used initially had a value of 0.1. This gave us an accuracy of 49.09747%. We then incrementally increased its value by 0.1 and discovered that the accuracy of the model increased as we approached 1. At $\alpha = 1$, the models accuracy was 51.98555%. Despite α being relatively large, it did not produce a sub-optimal set of weights.



Logistic regression for aplha = 0.1.

α = 0.1



Logistic regression for aplha = 1.0.

α = 1

4 Discussion of Results

The **Gaussian Naïve Bayes** algorithm performed sub-optimally compared to **Logistic Regression** but a lot better than **Discrete Naïve Bayes**. As stated above, a possible reason for the poor performance of **Gaussian Naïve Bayes** is the lack of data or the fact that some of the extracted features did not correlate sufficiently.
The **Discrete Naïve Bayes** algorithm performed the worst by far. This is most likely due to the limited number of data points and the lack of readily available features in the MIDI files.

4.1 Best Possible Performance

Logistic Regression was the best performer out of the 3 algorithms used. The difference in accuracy was approximately 15%. The main possible reason for the algorithm performing better is due to the lack of bias and high variance within our dataset and the algorithms itself. **Logistic Regression** was also less computationality heavy compared to the other algorithms which means it ran a lot quicker.
In conclusion, **Logistic Regression** performed better than complete randomness. Thus we were able to predict if certain musical pieces were composed by **Beethoven**, **Chopin**, **Mozart** and **Schubert** using the extracted features.

4.2 Recommendations to Others Working on This Data

- Don't use the raw MIDI data. Extracting established audio features is much more effective.
- Split the compositions into short snippets to increase the size of the dataset.
- Avoid audio features that give a huge number of values such as **Power Spectrum** since these will only add to *the curse of dimensionality*.
- Make sure to use MFCC as a feature, it seems to be a strong predictor.
- Due to the similarity between composers, this made it more difficult for our alrgorithms to properly identify the musical compositions. Therefore, choosing composers from different eras of classifical music could possibly give a higher accuracy.

5 Appendix A


```
<section start="16.384" stop="32.7679375">         <feature>
<name>Spectral Flux</name>             <v>8.51E-6</v>         </feature>
<feature>             <name>Compactness</name>            <v>7.406E5</v>
</feature>         <feature>             <name>Spectral Variability</name>
<v>8.891E-6</v>         </feature>         <feature>             <name>Root Mean
Square</name>             <v>1.367E-2</v>         </feature>         <feature>
<name>Zero Crossings</name>             <v>1.631E4</v>         </feature>
<feature>             <name>Strongest Frequency Via Zero Crossings</name>
<v>4.976E2</v>         </feature>         <feature>             <name>Strongest
Frequency Via Spectral Centroid</name>             <v>5.82E2</v>         </feature>
<feature>             <name>Strongest Frequency Via FFT Maximum</name>
<v>6.594E2</v>         </feature>         <feature>             <name>MFCC</name>
<v>-1.225E2</v>             <v>1.344E1</v>             <v>-1.062E1</v>
<v>-3.897E0</v>             <v>-3.706E0</v>             <v>1.668E0</v>
<v>2.414E0</v>             <v>2.589E0</v>             <v>-1.625E0</v>
```

<v>3.015E-1</v> <v>1.797E0</v> <v>1.988E0</v>
<v>-1.681E-1</v> </feature> <feature> <name>LPC</name>
<v>-9.465E-1</v> <v>9.681E-1</v> <v>-5.564E-1</v>
<v>5.075E-1</v> <v>-3.432E-1</v> <v>3.585E-1</v>
<v>-1.659E-1</v> <v>-2.825E-2</v> <v>-3.143E-2</v>
<v>0E0</v> </feature> <feature> <name>Method
of Moments</name> <v>2.216E-1</v> <v>1.208E4</v>
<v>1.11E8</v> <v>4.405E12</v> <v>3.371E17</v>
</feature> <feature> <name>Partial Based Spectral
Centroid</name> <v>4.152E1</v> </feature> <feature>
<name>Partial Based Spectral Flux</name> <v>-6.508E-3</v>
</feature> <feature> <name>Peak Based Spectral
Smoothness</name> <v>4.624E2</v> </feature> <feature>
<name>Relative Difference Function</name> <v>-4.923E0</v>
</feature> </section>