2018

Exploring the Features to Classify the Musical Period of Western Classical Music

Arturo Martínez Gallardo
Technological University Dublin

Recommended Citation

Gallardo, Arturo Martínez (2018). *Exploring the features to classify the musical period of western classical music.* Masters dissertation, DIT, 2018.

Exploring the features to classify the musical period of Western classical music

Arturo Martínez Gallardo

A dissertation submitted in partial fulfilment of the requirements of Dublin Institute of Technology for the degree of

M.Sc. in Computing (Data Analytics)

January 2018

DECLARATION

Signed: Arturo Martínez Gallardo

Date: 29 January 2018

ABSTRACT

Music Information Retrieval (MIR) focuses on extracting meaningful information from music content. MIR is a growing field of research with many applications such as music recommendation systems, fingerprinting, query-by-humming or music genre classification. This study aims to classify the styles of Western classical music, as this has not been explored to a great extent by MIR. In particular, this research will evaluate the impact of different music characteristics on identifying the musical period of Baroque, Classical, Romantic and Modern. In order to easily extract features related to music theory, symbolic representation or music scores were used, instead of audio format.

A collection of 870 Western classical music piano scores was downloaded from different sources such as KernScore library (humdrum format) or the Musescore community (MusicXML format). Several global features were constructed by parsing the files and accessing the symbolic information, including notes and duration. These features include melodic intervals, chord types, pitch and rhythm histograms and were based on previous studies and music theory research. Using a radial kernel support vector machine algorithm, different classification models were created to analyse the contribution of the main musical properties: rhythm, pitch, harmony and melody.

The study findings revealed that the harmony features were significant predictors of the music styles. The research also confirmed that the musical styles evolved gradually and that the changes in the tonal system through the years, appeared to be the most significant change to identify the styles. This is consistent with the findings of other researchers. The overall accuracy of the model using all the available features achieved an accuracy of 84.3%. It was found that of the four periods studied, it was most difficult to classify music from the Modern period.

Key words: MIR, Supervised machine learning, music genre classification, symbolic representation, Western classical music, Support Vector Machine

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor **Andrea Curley**, for her support, direction and guidance during the development of this dissertation. Her musical knowledge was particularly helpful.

Thank you to all my lecturers at DIT. I learnt a lot from each of them and the knowledge they passed on to me was vital to complete this work.

Finally and especially, I would like to express my deep gratitude to my lovely wife, **Áine**. I would like to thank her for her patience, help and support when I most needed it.

This dissertation is dedicated to the memory of my father, who passed away just weeks before I started this course. His continuous listening to classical music, developed my enthusiasm for music. I only wish I could have shared and discussed the process and findings of this thesis with him.

TABLE OF CONTENTS

Declaration .... **i**

Abstract .... **ii**

Acknowledgements .... **iii**

Table of contents .... **iv**

Table of figures .... **vii**

Table of tables .... **viii**

1. Introduction **1**

1.1 Background 1

1.2 Research problem 1

1.3 Research objectives 2

1.4 Research methodology 3

1.5 Scope and limitations 3

1.6 Document Outline 4

2. Literature review **6**

2.1 Introduction 6

2.2 What is MIR 6

2.3 Music data representations 7

2.3.1 Audio formats 7

2.3.2 Symbolic notation formats 8

2.4 Music score notation system fundamentals 9

2.5 Music genre classification 12

2.5.1 Feature extraction 12

2.5.2 Machine learning algorithms 15

2.5.3 Variable ranking 16

2.6 Classical Music time period classification 16

2.7 Conclusion 17

3. Design of the experiment 19

3.1 Introduction 19

3.2 Design overview 19

3.3 Data collection 20

3.3.1 Music formats used 20

3.3.2 Score collection process 21

3.3.3 Labelling the dataset 22

3.3.4 Dataset overview 23

3.4 Data preparation 24

3.4.1 Feature extraction 24

3.4.2 Data cleaning 32

3.4.3 Data transformation 34

3.5 Modelling 36

3.5.1 Group features in categories 36

3.5.2 Variable ranking 37

3.5.3 Support vector machine classifier 38

3.6 Evaluation 39

3.6.1 K-fold cross validation 39

3.6.2 Confusion matrix 40

3.6.3 Evaluation metrics 41

3.7 Conclusion 42

4. Implementation and results 44

4.1 Introduction 44

4.2 Data Collection 44

4.2.1 Music files 44

4.2.2 Labels 45

4.3 Data preparation 45

4.3.1 Feature extraction 46

4.3.2 Data cleaning 48

4.3.3 Data transformation *50*

4.4 Modelling 53

4.4.1 Variable ranking *53*

4.4.2 SVM predictive model *59*

4.5 Evaluation 59

4.6 Conclusion 60

5. Evaluation 61

5.1 Introduction 61

5.2 Classification results 61

5.3 Summary of findings 68

6. Conclusion 70

6.1 Research overview 70

6.2 Contributions 70

6.3 Limitations of the research 71

6.4 Future work and recommendations 72

Bibliography 74

Appendix A: Number of scores by composer 80

Appendix B: Summary statistics 81

Appendix C: Set of short music scores used for feature construction testing 83

Appendix D: Python implementation 85

TABLE OF FIGURES

Figure 2.1: Audio sample – Beginning of the Rondo alla Turca by W.A. Mozart......... 8

Figure 2.2: Sheet music score - Beginning of Rondo alla Turca by W.A. Mozart......... 8

Figure 2.3: Most important symbols in a music score... 10

Figure 2.4: Note and rest values 11

Figure 3.1: High-level design of the experiment... 19

Figure 3.2: Example of Humdrum format 20

Figure 3.3: Example of MusicXML format.. 21

Figure 3.4: Number of scores collected per period.. 23

Figure 3.5: Melodic simple intervals. Common name and number of steps below 29

Figure 3.6: Most common chord types, based on the note C 31

Figure 3.7: K-fold cross validation method... 40

Figure 4.1: Example of features with outliers 49

Figure 4.2: Scatter plot of two features before and after normalisation 52

Figure 4.3: Scatter plot of the normalised features in the new scale (0,1) 53

Figure 4.4:Variable ranking with the top five attributes for each feature group 54

Figure 4.5: Distribution of most significant rhythm feature values............................ 55

Figure 4.6: Distribution of most significant pitch feature values 56

Figure 4.7: Distribution of harmony feature values in number of chord notes 57

Figure 4.8: Distribution of harmony feature values in chord type formation............... 57

Figure 4.9: Distribution of most significant melody feature values 58

TABLE OF TABLES

Table 2.1: Music time periods and year ranges 17

Table 3.1: Main works collected 24

Table 3.2: Rhythm attributes 27

Table 3.3: Pitch classes and corresponding values 28

Table 3.4: Pitch attributes 28

Table 3.5: Melodic attributes 30

Table 3.6: Harmony attributes 32

Table 3.7: List of classifier models built with the number of features 37

Table 3.8: Confusion matrix for a binary classification 40

Table 3.9: Structure of a confusion matrix for the musical period classification 41

Table 3.10: Example of recall and precision metrics 42

Table 4.1: Example of initial dataset before feature extraction 46

Table 4.2: List of numeric features with low cardinality 51

Table 5.1: Model Accuracy list 61

Table 5.2: Confusion matrix of the rhythm features model 62

Table 5.3: Confusion matrix of the pitch features model 62

Table 5.4: Confusion matrix of the harmony features model 63

Table 5.5: Confusion matrix of the melody features model 63

Table 5.6: Recall table for each model and class (musical period) 64

Table 5.7: Precision table per class (musical period) 65

Table 5.8: Confusion matrix of the model using all the features 66

Table 5.9: Accuracy and precision per class with all the features 66

Table 5.10: Accuracy of the models to predict the composer period. 67

1. INTRODUCTION

1.1 Background

How does the human brain recognise and classify music? On the one hand, it may be considered a simple process, given that it has been shown that children as young as 9 months old can identify music as ‘happy’ or sad’ (Flom, Gentile, & Pick, 2008). Furthermore, adults with no formal musical training can classify a piece of music as belonging to a certain genre or style, simply by exposure and repeated listening to a wide range of musical pieces. However, this process is still not well understood. What specific features of music allow the brain to discriminate and classify a piece of music? Neuroscientists are analysing how different parts of the brain respond to different musical features (Jackendoff & Lerdahl, 2006). Musicologists are also concerned with how we perceive music. They aim to determine what particular musical qualities define a certain musical style.

The analysis of music information is a growing area of interest for researchers. Nowadays, the researcher has access to an extensive amount of music and they can use the latest advances in machine learning algorithms to extract knowledge and insights from music content.

A supervised machine learning model can learn, from a labelled set of music content, the relationships between the variables for each of the music styles. Then, the classifier can be tested on another unseen set by comparing the predictions made with the actual classes.

Identifying attributes that are particular to a music style or composer has many applications. For example, this knowledge can be used, along with computer music generation algorithms, to create music pieces that sound similar to a style or composer, such as the system DeepBach, that creates chorales in the style of Bach, and is highly convincing when tested with music experts. (Hadjeres, Pachet, & Nielsen, 2016).

1.2 Research problem

This study intends to explore the impact that different musical properties have in defining the music styles of Western classical music. Many works have used audio format as the main source to create genre classification systems with good performance (Sturm, 2012; Tzanetakis & Cook, 2002). This research has relied on the analysis of the auditory features of music, with signal processing and transcription system methods. However, they have not provided insights into how more complex musical features/concepts such as tonality, melody or harmony can allow for categorisation of music into different genres. An easier way to determine these musical properties, rather than using audio signal processing, is accessing the music scores directly.

The research question of the dissertation is:

What music properties are the most important in order to classify a classical music piece by its musical period?

In order to answer the question, this research project aims to extract features from music content and, use these to train a classification algorithm model and assess its predictive power to classify the music by music period or style.

The research will evaluate and compare the predictive capability of four main properties of music: rhythm, pitch, harmony and melody. It will also identify the most important specific attributes for each of these categories.

1.3 Research objectives

The main research objectives of this dissertation are :

- Investigate the current research conducted to classify music by genre.
- Review music theory and related literature that is necessary to understand the features that will be used.
- Download a collection of Classical music pieces that is appropriate to carry out the experiment and label each of these instances with the musical period according to their composer.
- Using a review of music theory and domain knowledge of the author, extract features from music content that can be used for music analysis.
- Use variable ranking methods to identify the attributes that contribute the most to determine the time period of a music piece.
- Train the classification models using different sets of related features, each of them related to a music property, and compare the results of the classification models to determine the most important features.

- Critically evaluate the overall findings and compare to what is known in the literature.
- Identify the limitations of the research and make recommendations for future work in this experiment and in the area.

1.4 Research methodology

The first step of this dissertation was to review the different representation and formats that are used to store music content, and to survey the research reported previously on musical genre classification. Following this, it was necessary to review music theory, so as to define the data that will be extracted from the files. Data preparation of these features was applied, including fixing data issues, handling categorical variables, creating derived features and transforming the data using a normalisation method.

Once the features are in a suitable form, the whole dataset will be divided into four, one for each music category that will be evaluated in the study (rhythm, pitch, harmony and melody). Then, two processes will be used to help answering the research question. Firstly, using the feature importance after applying a Random Forest algorithm in each of groups will be used to identify the specific attributes that most contribute in the classification. Finally, the classification models for each of the groups will be trained using the SVM algorithm, and evaluated with a k-fold cross validation method. The original dataset using all the features available, will be used to report the best results that the proposed features can get.

1.5 Scope and limitations

The scope of the study involves the extraction of a basic and limited number of features from the music scores. This will serve as a starting point to determine the most important differences in the music styles, therefore, most of them may already be obvious to music theorists.

It is beyond the scope of the research to perform a deep analysis of the music pieces to obtain very specific details that could discriminate small changes in the style. This type of analysis would require a profound review of music analysis studies and a highly proficient knowledge of the scripting language used.

This study will be limited to the music scores that are currently available. Using a much larger number of scores, it may be possible to determine with more precision the factors to classify the scores, which may have been missed due to the limited sample of scores obtained. The choice of music content will be limited to music composed for piano/keyboard instrument, in an effort to make the data sample the most representative without bias on the instrument particularities.

1.6 Document Outline

Chapter 2: This chapter gives a review of the literature that was needed to conduct this dissertation. It will start with a review to identify the relevant studies in the field of Music Information Retrieval and in particular for the purpose of classification tasks. This will be followed by a short section about the music theory necessary to understand the data analysed in this experiment. The review will also include the different attempts made to extract features from music content in any of the different sources, focusing on symbolic format (a music score). There will be a subsection to identify the different machine learning algorithms commonly used in classification tasks for music genres. Finally, the relevant research performed in the specific music category of the dissertation, Western classical music, will be reviewed.

Chapter 3: This chapter describes the design and methodology that will be followed in order to achieve the objectives of this research. Details on how the dataset will be collected and the features constructed will be included. The data cleaning and transformation that is planned to be applied before creating the models will be discussed. Finally the machine learning algorithm and settings, and the evaluation methods used to assess the models will be described.

Chapter 4: This chapter describes in detail the implementation and actual work performed in this experiment. Starting with the data collection of music scores, it will then describe the development of the feature extractors to create a set of attributes to carry out the experiment. The chapter will also include the issues found with the data and the preparation . It will include and discuss the first findings after data analysis and finally, the model implementation and details about how to create the evaluation metrics will be described.

Chapter 5: this presents the results of the classification models and critically evaluates the performance of each of them. The chapter also summarise the main findings of the whole experiment.

Chapter 6: The dissertation finishes with the conclusion chapter, giving an overall review of the experiment carried out and highlighting the main contributions to the field of music information retrieval. The limitations of the research will be pointed out and the chapter concludes with a discussion of recommendations for future work.

2. LITERATURE REVIEW

2.1 Introduction

In this chapter there will be a review of the different fields of research of significance for this dissertation. It starts with a brief presentation of what MIR is and its applications. This is followed by a discussion of the different representations of music content, audio format and symbolic format, the last being the focus of this study. Then the review will identify the different efforts made in music genre classification tasks. These studies will be divided between extracting features from music content and its relation to music theory and the different algorithms used for music classification systems. Finally, the review will be linked to the research question to identify the music classification studies in Western Classical Music and in particular to classify the work by its time period.

2.2 What is MIR

MIR stands for *Music Information Retrieval* and it covers the development of approaches to extract and analyse information from music. MIR involves many applications and it has been a growing field of research in the recent years with the rapid development of computer systems and algorithms to find patterns. (Typke, Wiering, & Veltkamp, 2005)

One of the applications is computer music generation. The extraction and analysis of music data can be used to create algorithms for automatic composition, for example, using harmonic rules and patterns learnt from a particular style. (Roig, Tardón, Barbancho, & Barbancho, 2014). MIR is also used for music recommendation systems. Extracting metadata and finding similar songs based on the audio content attributes (Kaminskas & Ricci, 2012), can add extra improvement to the traditional collaborative method based on user's ratings and playlists.

Another interesting application is the query by humming, by which the system takes a few seconds of a sung or hummed piece and retrieves a track from the database that matches the attributes of the input data (Pardo, Shifrin, & Birmingham, 2004). The principal challenge of such a system is that it cannot find an exact match using fingerprinting techniques because the song hummed is nowhere to be found in any database. For the moment only melodic and monophonic music is suitable for these type of systems (Kaminskas & Ricci, 2012).

In a deeper level of analysis, it can be used to detect plagiarism of a song by searching for melodic similarity (Grachten, Arcos, & Mántaras, 2004; Müllensiefen & Frieler, 2006) in a musical database.

Other MIR applications include automatic transcription, instrument recognition or lyric/audio synchronisation. Finally, as part of the process of indexing and cataloguing of music, features can be extracted to find patterns in the music and use them to classify works by musical genre, period, type, origin area or even composer.

2.3 Music data representations

Music information retrieval systems can be categorised depending on the *music content format* used to extract features. The data representation used to store music content can be divided into two main categories: *digital audio format* and *symbolic format* (music score). The features extracted and the techniques used depend greatly on the music data source.

2.3.1 Audio formats

Music content is usually represented as recorded audio; therefore the majority of work and MIR research done in feature extraction has been performed on audio tracks.

The raw audio files are normally in Waveform audio (WAV) or MPEG Audio Layer 3 (MP3) file extensions and contains the acoustic representation of the music using parameters such as amplitude and frequency. The researcher needs to use audio engineering and signal processing techniques to extract information from audio content. Other processes would need to be applied before features can be extracted, such as background noise reduction or equalization. In the paper of Tzanetakis and Cook, similarity analysis and classification of songs and pieces has been performed on audio tracks (Tzanetakis & Cook, 2002). As another example using audio content, mood and expression is automatically detected in order to add this metadata information to the songs and help the listener to choose a playlist depending on the mood (Lu, Liu, & Zhang, 2006)

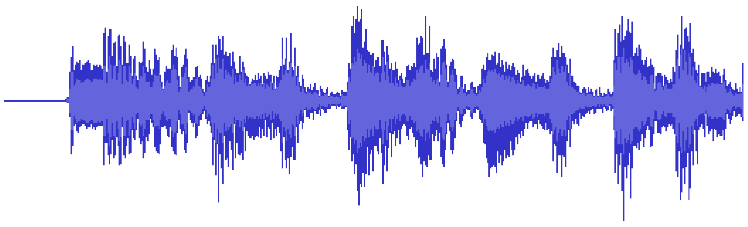


Figure 2.1: Audio sample – Beginning of the Rondo alla Turca by W.A. Mozart

2.3.2 Symbolic notation formats

The symbolic format is a close representation of the music score, which allows the composer to represent, store, and publish the music they create and to make it available for performers to play. A music score contains a different set of symbols indicating the duration of the notes, the presence of rests, how high or low a note sounds, tempo and also dynamics and expression information. A music score includes more high-level features that enables a music theorist to perform analysis using their knowledge of the domain.

Despite the extensive research using audio content, this representation of music has some drawbacks. Audio files are relatively large and the process to extract audio descriptors involves a high computational effort. Audio samples don't have explicit information about notes, voices, tempo or rhythm that researchers from music theory communities can understand and analyse to perform their studies. (Ponce de León, Iñesta, & Rizo, 2008).

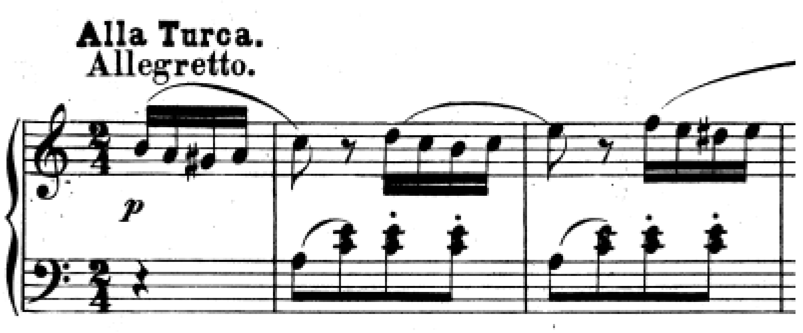


Figure 2.2: Sheet music score - Beginning of Rondo alla Turca by W.A. Mozart

One of the formats which is more commonly available and has been used in many studies, is the *MIDI* format. MIDI (Musical Instrument Digital Interface) files, can store instructions that can be sent to synthesisers or sound cards in order to play a music piece. This format is really popular and acts as a hybrid between an audio file and a purely symbolic notation format. However, in a MIDI file not all of the information from a music score can be represented (Good, 2001).

Other formats more closely related to the digitally published music scores are the ones that music notation software can produce. Professional software such as Finale or Sibelius or open source ones such as MuseScore, can work with *MusicXML* files. A MusicXML file is a plain text XML-based file, with a hierarchical structure of tags, which includes information about key signature, tempo, notes, dynamics and expressions (Good, 2001). The book *Beyond MIDI* (Huron, 1997), describes other efforts to create other formats to encode music score information to be used for analysis, for example, the *humdrum* format that has a large catalogue of music.

In order to extract information from these symbolic formats, parsers can be created to run through the text files and extract the relevant information, for example the python library music21 (Cuthbert & Ariza, 2010), that can be used to get the information from music scores.

2.4 Music score notation system fundamentals

The difficulty in memorizing or showing how a piece of music should be played gave rise to the development of music notation systems. Systems to secure and transmit music have been used since immemorial times. These systems evolved over time and were adapted to the needs of composers. The current notation system used in Western music has not changed remarkably since the 17th century (Fradera, 2003)

Musical notation uses a series of symbols and marks to show the duration and pitch of the sounds. There are symbols to express dynamics, tempo and articulation as well. To some extent they can include text in the form of lyrics or to give the performer more indications on how the music should be played.

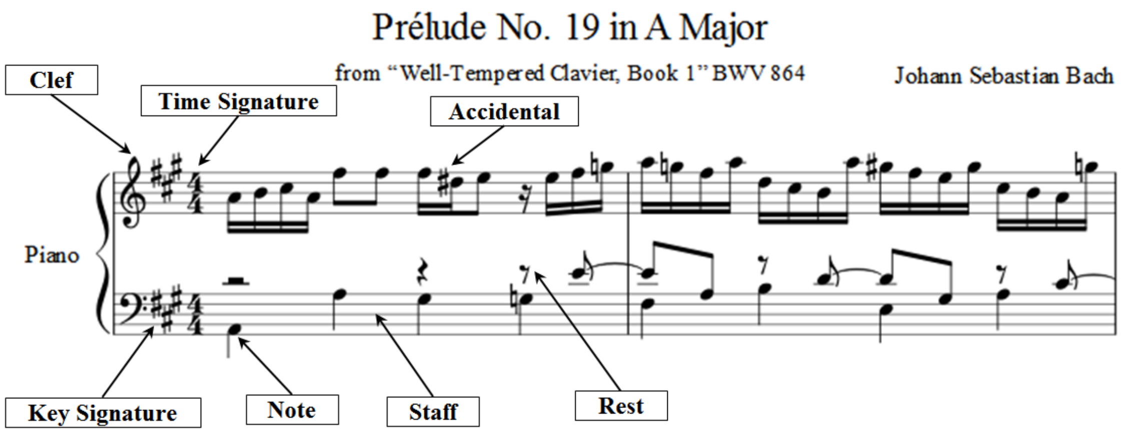


Figure 2.3: Most important symbols in a music score

The notation system allows the addition of a large number of symbols and marks but the most important are shown in Figure 2.3. The ***staff*** is where the rest of the symbols are placed so every score includes at least one. If the composition is for more than one instrument there will be a staff per instrument. In the case of the piano, it is common to use two staves, usually the upper one for the right hand and the lower one for the left hand.

The ***notes*** are placed in the staff and the position of a note will determine the ***pitch*** of the sound. If more than one note is placed in the same position vertically, they will be played at the same time. The pitch of the note is not absolute and it depends on the ***clef***. A note placed in the same position could have a different pitch. The clef defines the range of notes used in that particular staff. In the example shown, the first staff has a G clef and it is the most common in modern notation. It indicates that a note placed on the second line from the bottom is a G and the rest of notes in the same staff will have that reference. The second staff has an F clef, in this case the second line from top indicate an F. These combination of G and F clefs (also called Treble and Bass clefs) are the most common used in piano.

Accidentals can be placed just beside the note and they will alter the pitch of any of the seven natural notes (C, D, E, F, G, A, B). A sharp ♯ symbol will raise the pitch by a semitone and a flat ♭ symbol will lower it by a semitone. A natural symbol ♮will cancel the previous accidental on that note. The ***key signature*** represents the tonal key of the music in the score. In the above example, the piece is written in A major, meaning that there are three accidentals specific for that key (F♯ , C♯ and G♯). This simplifies the notation so there is no need to use accidental marks when these notes appear in the score. It is placed at the beginning of the score and it will be effective during all the piece or until a key signature change.

The duration of each note is indicated by using different shapes of note heads and stems, and it will determine the *rhythm* of the score. These durations are not absolute, and only proportional to the rest of the notes. Unless there is a ***tempo*** indication in the score, the absolute duration is a choice of the performer.

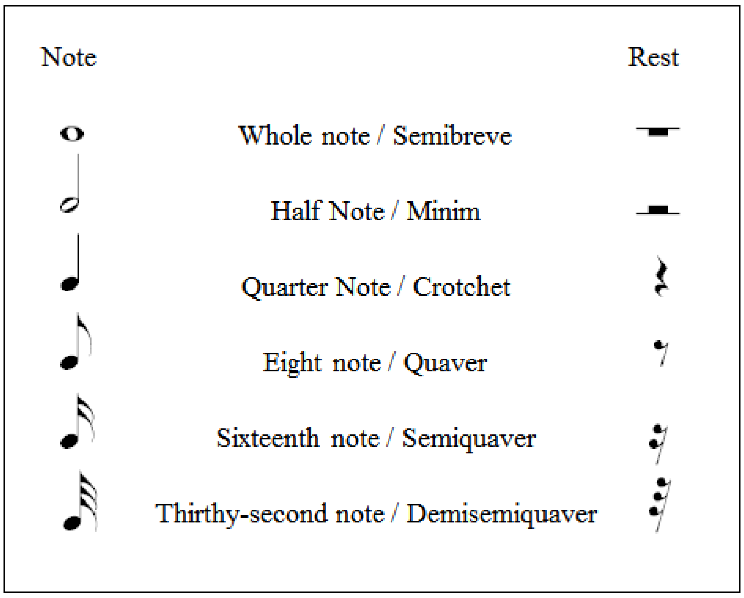


Figure 2.4: Note and rest values with American and British English spellings

Figure 2.4 shows the most common note values and the corresponding rests. A dot can be added to the note and it will increase the note duration by one half. More dots can be added and will alter the length in another one half of the previous duration. The ***time signature*** will define the meter of the score. It is an abstract concept that will indicate the number of beats in the score and the total length in each uniform section or measure. The numerator defines the number of notes in each measure of the value defined in the denominator. For example, 3/4 means that each measure has 3 crotchet/quarter note beats. The time signature will determine the rhythm of the piece. In a 3/4, usually the first beat is strongly accented, while the other two are weak. This triple time is common in waltz, minuet or mazurka styles (Taylor, 1991).

There are many other symbols and marks that can be used in the notation system and this is only a brief introduction in order to understand the data that will be managed for the experiment.

2.5 Music genre classification

One of the most important fields of the Music Information Retrieval (MIR) is music classification. It has many applications such as cataloguing large collections of music or giving music recommendations for online shops and streaming services. There are a number of issues related to classifying music into different genres, for example the difficulty in clearly defining the genres when the genres overlap or are not discriminatory (McKay & Fujinaga, 2006). Despite these issues, end users are still keen to browse by genre from a musical database, based on the results of a survey on music information seeking behaviours (Lee & Downie, 2004)

Any process of music classification consists of feature extraction from the data sources and the creation of a classifier using machine learning algorithms. (Corrêa & Rodrigues, 2016)

2.5.1 Feature extraction

Before starting any classification task, features need to be extracted from music content. Once the features have been created from the different music sources, they can be used for multiple types of analysis. Using pattern matching techniques and classification algorithms, these features can help indexing, recovering and classifying musical content.

Features extracted from audio

In a survey of music genre recognition, Bob Sturm points out that 79% of experimental work was done using audio music (Sturm, 2012), as this media is more interesting for the general public.

Features extracted from audio are considered low-level features. For example, *timbre* is one the features that can be easily extracted from audio, with the use of fast Fourier transform (FFT) techniques (Fu, Lu, Ting, & Zhang, 2011). Timbre relates to the quality of sound, and it will vary for different types of sound sources, such as different instruments, even when playing the same note. Up to 2006, features related to timbre are the ones that have been used for most classification tasks and the performance has been limited. (McKay & Fujinaga, 2006). There was a need to add high-level musical abstractions and cultural features in order to improve performance and success rates.

The extraction of more high-level features such as *rhythm* or *pitch*, although possible, is a difficult task when using raw audio signals. The study of Tzanetakis et al., evaluated the accuracy of extracting the pitch histogram (global census of the pitch content) of musical pieces from audio form. The researchers compared the music genre classification results when using pitch features from both audio and symbolic forms of the same songs (Tzanetakis, Ermolinskyi, & Cook, 2003), and obtained better performance using the features extracted from symbolic data (MIDI).

Conversion from audio to symbolic format

Another area of research has been focused on using audio content as initial source but converting it to symbolic format. In order to be able to use these high-level features in the research, some studies have been performed to be able to automatically transcribe audio data into symbolic representation (Lidy, Rauber, Pertusa, & Iñesta, 2007). The results indicate that there is still room for improvement in the accuracy of the transcription systems, particularly to capture rhythm and harmonic features or information other than pitches and note durations. Another implication is that these systems still need, at a certain level, some manual input annotation and it becomes impractical for large datasets. As the transcription systems improve, researchers will focus more on high-level features using audio.

Features extracted from symbolic format (music scores)

Studies have been performed in the area of music classification, applying different music descriptors extracted from the symbolic representation of the songs.

In some papers, the research focuses on ***rhythm*** attributes. Rhythm is related to the duration of the notes and its combination in time/beats. For example, different representations of rhythm patterns were extracted and compared to prove that rhythm alone can be used to perform music similarity queries. (Lev et al., 2011). Note duration histograms were used in (Karydis, 2006) to classify five different musical styles (ballad, choral, fugue, mazurka and sonata) using a k-NN algorithm.

Other papers use ***pitch*** descriptors as the main attribute for the classification. Pitch is the quality of a sound in terms of how low or high is. The frequency of each different note appearing in the music, also called pitch histograms, has been used widely in the literature, for example to classify music from classic, jazz and pop style (Tzanetakis et al., 2003).

Other studies use ***harmonic*** features, which are the features that can be extracted from notes that are played at the same time. Chords and complex harmonic features based on tonal theory are used to inspect the style of three different composers, Mozart, Schubert and Brahms (Ferkova, Ždimal, & Šidlik, 2007). Studies using harmonic features are less explored in the literature (Corrêa & Rodrigues, 2016)

Melodic features are also considered in the literature. Melody is a sequence of notes with their durations and pitches in horizontal movement, as opposed to the harmony, where it is considered vertical. Some papers use melodic intervals as part of the feature vector to perform classification or to obtain the similarity between songs (Müllensiefen & Frieler, 2006; Ponce de Léon & Carlos, 2004).

The paper of McKay and Fujinaga, presented a system called jSymbolic, which could be used to extract a large variety of musical features from MIDI files (Mckay & Fujinaga, 2006). Most of the features are original and others are based on previous research and music theory studies. They divided the features in seven categories: instrumentation, texture, rhythm, dynamics, pitch statistics, melody and chords. The system was then tested to perform classification of 9 genres with success rates of 90%. Some of these features have been used in other research papers such as in (Hillewaere, Manderick, & Conklin, 2009) where 62 of these global features were used to classify folk songs by their region origin.

To conclude, the creation of the features presented in the studies, especially in the ones using music scores, take advantage of music theory analysis and musical knowledge, in order to understand the data that the researcher is managing. For example, the music theorist Leonard B. Meyer, compared music styles by looking for repeated patterns in rhythm and melody using statistical algorithms (Meyer, 1989). Allen Forte developed the concept of set theory by using pitch-class sets to analyse the harmony of tonal and atonal music (Forte, 1988). The relative frequency of use of the combination of these note class sets can determine the music style and tonality of a piece. Another popular style of music analysis is the Schenkerian analysis, in which it seeks to find the melodic or harmonic connections between consecutive notes or chords respectively. (Marsden, 2010)

2.5.2 Machine learning algorithms

The selection of features is a fundamental step in the creation of classification models. In particular, for music genre classification tasks, the choice of features is vital, while the classifier algorithm, despite being important, normally reflects the changes in the feature selection (Corrêa & Rodrigues, 2016; Karydis, 2006).

The comparison of results from different papers is a difficult matter as studies with the same datasets for symbolic music are not very common in the literature. The same classifiers can perform very differently with other datasets. Music Information Retrieval Evaluation eXchange (MIREX) is a community that runs a yearly competition to evaluate the efforts in many different MIR tasks. Audio genre classification task was run from 2005 until present (2017), however for symbolic genre classification, the contest only took place in 2005. The task had two sets of categories, one with 9 genres using 225 MIDI files and the other one with 38 genres, subgenres from the main 9 categories, and using 950 MIDI files. All participants had to use the same training and test splits. The winner achieved an accuracy of 84.4% for the 9 classes task, and 46.1% for the 38 genre classification task. A classifier ensemble of neural networks and k-NN was applied to the task using features to represent pitch, texture, rhythm, dynamics, melody and chords. (Mckay & Fujinaga, 2005; McKay & Fujinaga, 2006).

Almost every research uses a different dataset, however some studies have been testing different classifiers with the same set of files. A comparison of Naïve Bayes, Support Vector Machine (SVM), k-NN and discriminant analysis methodologies has been performed to classify classical music scores by composer (Lebar, Chang, & Yu, 2010), concluding that although some classifiers were better than others to distinguish some composers, SVM was consistently the most accurate.

SVM is an algorithm that looks to find an optimal hyperplane to separate samples from different classes (Cortes & Vapnik, 1995). It works well when there is a large number of attributes compared to the numbers of samples (Corrêa & Rodrigues, 2016; Xu, Maddage, & Shao, 2003).

2.5.3 Variable ranking

Some machine learning algorithms, including SVM, are considered a “black box” algorithm and the results are hard to interpret, especially if using a non-linear kernel method (Ben-Hur & Weston, 2010). The output doesn’t provide information about the importance of the features used to create the model. Other methods can be used to identify the variables that have better predictive power for the classification. As stated in the paper of (Guyon, Elisseeff, & De, 2003), these methods are divided into wrappers, filters and embedded.

- **Wrappers:** Assess the efficiency of multiple combinations of subset of variables by comparing the prediction performance using a machine learning algorithm. It is computationally intensive because it is a ‘brute force’ method.
- **Filters:** this method is independent from the machine learning algorithm, and instead of using accuracy/error rate in a subset, this method uses statistical tests, such as Pearson correlation, LDA, chi-square with the outcome variable. This method is generally used as a pre-processing step to discard features before feature selection.
- **Embedded:** combines wrapper and filters methods. This uses machine learning algorithms that have built-in methods to perform feature selection, according to specific criteria. This method is more efficient in several respects: they are faster and all the available data can be used, instead of splitting in training and testing. (Guyon et al., 2003). One of the most popular algorithms using this method is Random Forests (Genuer, Poggi, & Tuleau-Malot, 2011).

2.6 Classical Music time period classification

Some studies have been identified which include classical music for classification systems. For example, a research with a number of melodic, harmonic and rhythm descriptors is presented to classify monophonic music in the two different genres, jazz and classical music (León & Iñesta, 2004). The best results achieved in the study after feature selection is 88.7% average success using k-NN.

Regarding the field of classical music, one of the most extended classification levels is by composer. A system that can classify a composer, can be used to certify if a music score belonged to a certain artist. For example, a pattern recognition approach was used to analyse disputed organ works by Bach (Van Kranenburg, 2006).

Western classical music can be classified depending on their historical periods. These eras include Medieval, Renaissance, Baroque, Classical, Romantic, and Modern. Music historians don’t agree on the start and finish dates of the different periods, partly because the styles that correspond to these periods changed smoothly. During a number of years, two periods overlapped and were present side by side, depending on the composers or even geographical locations (Grout & Palisca, 1996). It is important to note that the musical periods are defined by the style in music composition rather than the exact time of the music. The following Table 2.1 shows the main periods of Western classical music with the approximate years.

Period Name	**Year Range**
Medieval	500-1400
Renaissance	1450-1600
Baroque	1600-1750
Classical	1750-1810
Romanticism	1810-1910
Modern	1900 - Present

Table 2.1: Music time periods and year ranges

Works that use exclusively classical music score sets to create classification systems are not numerous in the literature, especially for determining the musical period.

According to Weiss and Schaab, tonality and harmony play an important role in determining the musical style in Western classical music (Schaab & Weiss, 2015). In their work, using datasets of audio recordings, chroma-based vectors are created based on the audio signal spectrogram, to be used for the classification of periods.

2.7 Conclusion

Music Information Retrieval (MIR) has been a field of research for many years. Different formats to store music content and the features that can be extracted from them were examined in this review. After a detailed review of the literature and prior work in the area, the following conclusions can be identified:

Motivation of the research

Although there has been research performed on automatic classification systems, it has generally been by classifying top-level and broad genres such as Rock, Jazz, Classical, Rap, etc… To the best of my knowledge, researches involving classifying more complex music, such as Western classical music into their musical time periods have not been sufficiently explored using music scores, mainly due to the lack of data available for the task. This project will tackle the problem of classifying classical music scores by period.

Choice of data source format

Within the analysis of the different music representations, the symbolic format, and in particular MusicXML and humdrum, is the format that can capture a music score in as much detail as possible and would be the most suitable to analyse the music theory aspects that can help to determine the time period (era) of Western classical music. The use of audio content, although extensive in the literature, presents some drawbacks for this type of classification.

Attributes to work with

There is a large amount of attributes that can be created from music scores and used for classification projects. They can be categorised in different attributes such as pitch, rhythm, harmony, melody, timbre, dynamics and expression. These attributes are directly linked to music theory and analysis performed by musicologists.

Machine learning algorithm

Based on research into similar case studies, Support Vector Machine is the most common algorithm used to classify music works as it gives the best accuracies. Variable ranking and feature selection methods can be used to: identify the most important predictors for the classification, acquiring a better data understanding, and also used to create models with less features that could improve accuracy and reduce over-fitting.

3. DESIGN OF THE EXPERIMENT

3.1 Introduction

This chapter will describe the design and methodology used to carry out the experiment of this dissertation. It will start with a design overview diagram which illustrates the main steps that were followed. It will then discuss the need to build a dataset for the purpose of the thesis and the details of the data collection process required to create it. A description of the process to construct the features from the raw music scores will be provided. The features will be divided in different factors: pitch, rhythm, melody and harmony. Complete details of the features for each of the categories will be provided in this section. A section about overall data cleansing and transformation is included as well. The creation of the different models and the algorithm used will be described. Finally, the evaluation methods and the different metrics used to analyse the results will be discussed.

3.2 Design overview

The experiment performed in this dissertation has four main steps: data collection, data preparation, data modelling and evaluation. All the steps will be developed using Python scripts. The main scripts are shown in Appendix D. An overview of the stages and processes are shown in Figure 3.1 below.

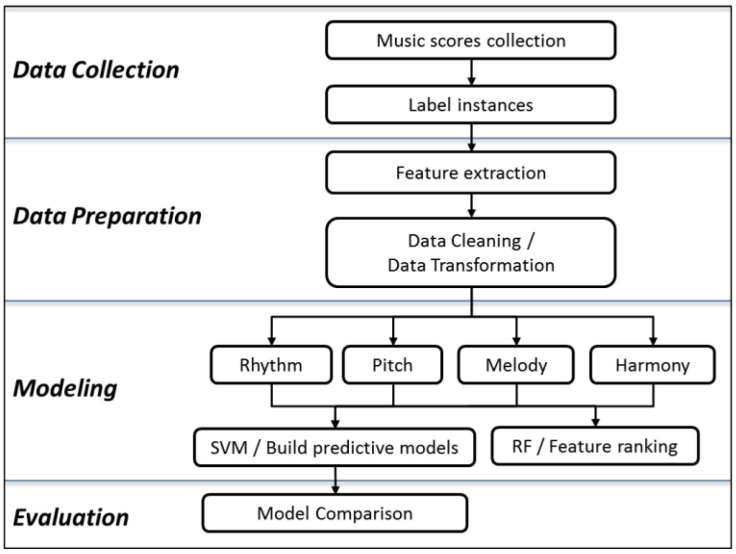


Figure 3.1: High-level design of the experiment

3.3 Data collection

A dataset will be built for the purpose of this dissertation due to the lack of resources available for this specific task, as mentioned in the literature review. In order to create this dataset, music scores will be downloaded from different sources. The information that is shown in a score can be represented in digital files. A piece of music can be stored as a digital image or in PDF format and printed. In order to be able to extract information of notes and durations from an image, complex Optical Music Recognition (OMR) software has to be used. These systems have improved in recent years but are still face many challenges, preventing the exploitation of their full potential (Bainbridge & Bell, 2001; Ringwalt, Dannenberg, & Russell, 2015). However, scores can be created with music notation software and exported to different formats. There are a number of these scores available to download in the web. Two formats were used for the data collection.

3.3.1 Music formats used

Humdrum format: Humdrum format is a robust format for music notation with a large catalogue available. It is a plain text format and it has been used for computational musical analysis in diverse applications. (Sapp, 2005). It can also contains metadata information using a specific syntax such as '!!!COM' at the start of the line to indicate the composer name as it is shown in the example in Figure 3.2.

Staff or parts of the score are organised in different columns. Notes played at the same time are on the same line of the text file. In the example, the first two measures of a piece by Mozart are shown in Humdrum format beside the original music score.

```
!!!COM: Mozart, Wolfgang Amadeus
!!!OTP: Viennese Sonatina No. 1 in C-major
**kern       **kern
*clefF4      *clefG2
*M4/4        *M4/4
=1           =1
2CC 2C       2c 2cc
2EE 2E       2e 2ee
=2           =2
2GG 2G       2g 2gg
4r           4r
8G’          8gg’
8G’          8gg’
```

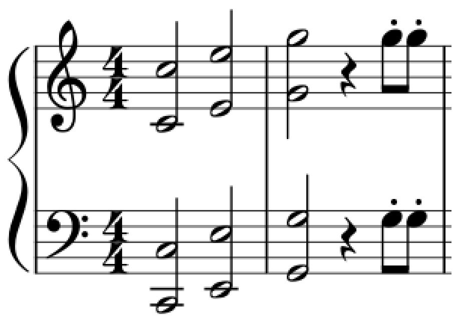


Figure 3.2: Example of Humdrum format

MusicXML format: MusicXML started as a way to have a common format that could be used in many notation software programs, apart from their native file format. The MusicXML uses a hierarchy system of XML-based tags to represent the content of sheet music (Good, 2001). There are tags with metadata information such as composer, work and movement.

```
<note>
       <pitch>
              <step>G</step>
              <octave>4</octave>
       </pitch>
       <duration>2</duration>
       <type>half</type>
</note>
```

Figure 3.3: Example of MusicXML format

An example of how the information related to a note is stored in a MusicXML file can be seen in Figure 3.3 above. In this example the note G is played in the fourth octave with the duration of a half note. The note tag will be part of a measure tag along with other notes. Accidentals, dynamics, expressions, slurs and other symbols that can be contained in a music score, can be represented with tags. Key and time signatures, clefs and tempo tags are also included at the start of the file.

3.3.2 Score collection process

Music files sources

The scores in the humdrum file format will be downloaded from the KernScores library (Sapp, 2005). Kern Scores is the main online database of scores in humdrum format and is accessible browsing by composer or title. The website includes zip files with entire collections by composer. This catalogue is extensive but it lacks music files from the modern period. That is why other sources will be used to obtain additional music files.

MusicXML scores will be downloaded mainly from the website musescore.com, a community system to publish and share scores online, created with the free MuseScore software. More modern period scores were found in this source to complement the catalogue obtained from the KernScores library. A python script will be written to automate the navigation through the website and download the files, as this website did not offer bulk downloads.

Even though two different formats were used to create the dataset collection, the information needed to create the features was available in both of them. The only difference is the representation of that information.

Considerations - Same instrument

In order to make the data sample as uniform as possible, all the works collected were for the same instrument. This was done to prevent certain attributes of the music that are particular to a specific instrument affecting the results of the experiment. For example, in flute and in general in wind instruments, it is not possible to play two notes at the same time. Unless there are a balanced number of pieces per instrument and per period, this would lead to a sample that is not representative.

Piano works have been used for the dataset because they allow the extraction of harmonic information in the form of chords as the piano is a polyphonic instrument. This way, the main characteristics of music such as pitch, rhythm, melody and harmony can be analysed, in the interest of determining the features most important for the classification. The piano is also a very popular instrument with a large catalogue of pieces composed for this instrument or related ones, such as harpsichord. Furthermore, there are works for piano in all the time periods of the study.

3.3.3 Labelling the dataset

For the purpose of the classification using supervised machine learning, each of the records of the dataset needs to have a label that identifies the row with a particular class. This way, a model can be trained with the information of the features and what class it represents. In the case of this study, the class is the music era. Although the music periods of classical music are disputed with regard to the exact dates of the year ranges (Rodriguez Zivic, Shifres, & Cecchi, 2013), it is common practice to identify a composer with their period. This method to classify the pieces has some drawbacks, as not all the pieces of the same composer share the same style of a period. This is particularly notable for authors that composed their work in the latter years of an era as this work can be influenced by the style of the next period. Composers could have changed their style during their lifetime. Most of the limitations in any type of classification systems come from the fact that the classes are not clearly defined and are subjective.

Therefore, using the composer that is included in the music files, the corresponding musical period will be added to the dataset.

3.3.4 Dataset overview

A total of 878 music files were finally collected to build the dataset. In the collection, the movements that are part of the same work were considered as different rows in the dataset. This is usual in the Sonata form, that contains three movements or parts. Each movement has its own identity, tempo and style and therefore, they were handled as individual records.

Because files were collected from different sources, the same score could have been downloaded more than once. Lists of music work titles by composer were created in order to facilitate the removal of duplicates from the dataset. The final dataset had a good representation of music from each of the periods. It was more difficult to get music from the modern period. This might be due to some modern pieces still being under copyright, as opposed to works from Bach that are public domain nowadays. The final number of pieces collected per musical period are shown in Figure 3.4 and the breakdown by composer is shown in the Appendix A.

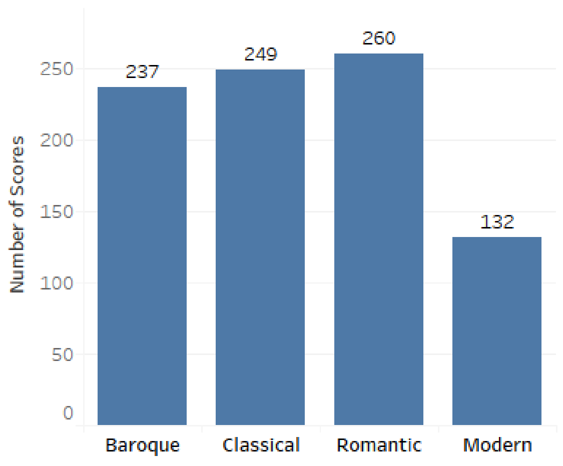


Figure 3.4: Number of scores collected per period

A summary of the most important piano scores collected are shown in Table 3.1

Period	**Works Collected**
Baroque	- Well-Tempered Clavier I and II by Bach (48 preludes and 48 fugues) - Golberg Variations by Bach (Theme and 30 variations) - 15 Two-part Inventions by Bach - 59 Sonatas by Scarlatti - Various pieces by Buxtehude, Haendel, Purcell and Rameau
Classical	- 17 Sonatas by Mozart(51 movements) - Other pieces by Mozart including Variations, Fantasies, Minuets, Divertimentos and Rondos. - All the 32 Sonatas by Beethoven (102 movements) - Other pieces by Beethoven such as Diabelli Variations, Preludes, Sonatinas and Rondo. - Sonatas by Haydn (17 movements) - 6 Sonatas by Clementi (17 movements)
Romantic	52 Mazurkas by Chopin 24 Preludes by Chopin Several Waltzes, Nocturnes and Etudes by Chopin 25 Preludes by Alkan Trascendental Etudes By Liszt German Dances and Ecossaises by Schubert Six fugues by Saint-Saens 24 Preludes by Hummel Individual pieces from Schumann, Glinka, Mendelsshon, MacDowell, Grieg, Tchaikovsky and Mussorgsky
Modern	Pieces by Debussy including Preludes, Arabesques and Children’s corner Gnossiennes and Gymnopedies by Satie 14 Preludes and some Etudes by Rachmaninoff Individual pieces from composers such as Ravel, Prokofiev, Sibelius and Poulenc

Table 3.1: Main works collected

3.4 Data preparation

Data preparation is an important step for any classification task or in general, any data mining system. In this experiment, it starts by constructing the features from the music scores raw files. After that, issues have to be identified because data with low quality will not give the appropriate results. This can be done by exploring the data to point out the different issues, such as missing data, outliers and inconsistencies. It will follow by choosing the approach to take in order to fix or at least minimise the impact of the problems. Data transformations to prepare to data to a suitable form will be discussed too.

3.4.1 Feature extraction

After the dataset has been built, the next step is to extract attributes from each of the music scores. The extraction of these features is a very important step and the foundation of any classification task. Without a good set of features that can explain the different aspects of a song, there is little that can be done even with an excellent and optimised classifier algorithm (Corrêa & Rodrigues, 2016)

There is no defined set of features that must be extracted from music scores. Depending on the research, the feature extraction can be focused on different aspects of music. For example, for an algorithm to divide a song into musical phrases, a complex set of features based on dynamics and beats was used (Thom, Spevak, & Höthker, 2002)

In this study, the goal was to classify pieces by music period using different types of attributes from the music score. A wide range of features from many different characteristics of music were used as a way to cover all the styles and aspects of the musical period from a range of over 300 years. The majority of attributes are global to the pieces, such as minimum, maximum values, average and histograms, and there is no profound analysis of patterns and motifs. Depending on the results and for future work, more time could be dedicated to explore more local features and particular factors in the structure of the score. Most of the features used in this study, were based on the works that were mentioned in the literature review, the main one being the research of McKay and Fujinaga (Mckay & Fujinaga, 2004).

Rhythm and pitch are the most basic categories and the two attributes that make up a piece of music.

- **Rhythm** relates to the horizontal movement, describing the length of the notes

- **Pitch** relates to the vertical, describing how high or low the sound is.

All the features extracted in this dissertation could be included in these two types. Towards investigating more specific factors, two more concepts were included in this thesis, melody and harmony.

- **Melody** is a combination of rhythm and pitch over time. In other words, a succession of notes with a particular duration and pitch.

- **Harmony** refers to the notes that are played at the same time and the pitch difference that will make different chords. The formation of these chords are related to the tonality, which is a factor that can differentiate musical genres.

The music scores files, which are plain text format, were parsed into objects containing all the notes information and their order within the song. These objects were then analysed in order to extract the features of each song.

Rhythm features

The rhythm is related to the duration of the sounds or notes in music and it is the most basic property in music. In fact, some instruments can only produce rhythm with no change in the pitch or frequency of the sound. Beat and tempo can also define the rhythmic information in a piece. As mentioned earlier, only global statistics about the rhythm are included in this category. Different changes in rhythm along the score will be related to the melody.

In order to extract the rhythm features of a single score, the following attributes were compiled for each note found in the score:

Duration: is the duration of the note in units of quarter/crotchet length. For example, a half note would have the value 2. A decimal value will be used to store the exact duration, if using dotted duration. A dotted quarter length note would be 1.5.
Duration name: is the name of the note, related to the duration but as string. This is used to compile the duration histogram.

Using the information from this list of notes, rhythm features can be created. In Table 3.2 a detailed description of all these features is shown. Other global information is obtained from the score to complete the rhythm category of features.
Information such as:

Rests**:** are the total number or rests or silences found in the score. The feature value will store the fraction of rests compared to notes.
Time signature: is the number of time signature changes and also the information about the initial time signature.
Grace notes**:** are the notes that are ornaments added to a note. These notes have a note duration close to 0, therefore, they will not be considered in the above list of notes. However, the proportion of grace notes will be stored.

Expressions**:** are notes that have an expression information linked to them, such as mordent, trill, tremolo, etc…

This information will be used to construct the rest of the features in Table 3.2

Attribute	Description
r_minimumDuration	Shortest note duration
r_maximumDuration	Longest note duration
r_noteDurationAverage	Average of the note duration. Sum of all note durations by number of notes
r_noteDurationStdDev	Standard deviation of all note durations
r_numberOfDistinctDurations	All the different note durations used in the score
r_mostCommonNoteDuration	Duration of the most common note in the score. The unit is the quarter note and has the value 1.0
r_mostCommonNotePresence	Percentage of use of the most common duration
r_twoMostCommonNotePresence	Percentage of use of the two most common duration
r_distanceTwoMostFrequent	Duration between the two most frequent note durations in times the duration is longer than the other
r_dh_32nd r_dh_16th r_dh_Eighth r_dh_Quarter r_dh_Half r_dh_Whole r_dh_Breve r_dh_Dotted_32nd r_dh_Dotted_16th r_dh_Dotted_Eighth r_dh_Dotted_Quarter r_dh_Dotted_Half r_dh_Dotted_Whole	Frequency of use of each of the notes in a histogram of note durations.
r_nonBasicDurationsPresence	Fraction of notes that are not in any of the previous bins.
r_timeSignatureChanges	Number of times there is a meter/time signature change in the score.
r_timeSignatureLength	Length in quarter notes of the main time signature.
r_dupleMeter	If the time signature is 2/2, 2/4, 4/4, 6/8
r_tripleMeter	If the initial time signature numerator is 3 or 9, three beats per measure
r_noteDensity	Number of notes per unit (quarter length). Total number of notes by total length of the score.
r_restPresence	Percentage of rests/silences used in the score
r_graceNotePresence	Fraction of notes that are grace notes (ornaments)
r_expressionsPresence	Fraction of notes that include expressions (mordente, tremolo, etc...)

Table 3.2: Rhythm attributes

Pitch features

Pitch determines how high or low a note sounds. In order to extract the pitch features of a single score, all the notes used were computed and a list was created with the following pitch information of each note:

Pitch number: is a unique number that identifies the pitch of the note. It represents the absolute pitch of the note. Instead of using frequency in hertz, it was simplified by using an integer number. Each note would have a unit difference with the next. This would represent the semitones. For example, C3 would be 48, C#3 = 49 and so on.

Pitch class: is the note without taking into account the octave the note is in. For example, the class of the notes C3 and C4 will be just C. The twelve possible classes and their values are shown in Table 3.3

Note	C	C#/D♭	D	D#/E♭	E	F	F#/G♭	G	G#/A♭	A	A#/B♭	B
Value	0	1	2	3	4	5	6	7	8	9	10	11

Table 3.3: Pitch classes and corresponding values

Pitch Accidental: is a note that is not included in the key signature. For each note, a Boolean value will be stored representing if the note is an accidental or not.

Using the information of this list of notes, pitch features can be created, for example pitch histograms, that have been widely used in the literature (Tzanetakis et al., 2003). In Table 3.4, detailed description of all these features are shown. The number of key signature changes was also computed globally.

Attribute	**Description**
p_minimumPitch	The minimum absolute pitch note used
p_maximumPitch	The maximum absolute pitch note used
p_pitchAverage	The average of pitches
p_pitchStd	Standard Deviation of the pitches
p_numberOfPitchClasses	Number of different pitch classes used in the score (1-12)
p_numberOfPitches	Number of different absolute pitches used in the score
p_distanceStrongestPitches	The distance in semitones between the two most used pitches
p_distanceStrongestPitchClasses	Shortest distance in semitones between the two most used pitch classes.
p_ph01	Fraction of the 1st most used pitch class
p_ph02	Fraction of the 2nd most used pitch class
p_ph03	Fraction of the 3rd most used pitch class
p_ph04	Fraction of the 4th most used pitch class
p_ph05	Fraction of the 5th most used pitch class
p_ph06	Fraction of the 6th most used pitch class
p_ph07	Fraction of the 7th most used pitch class
p_ph08	Fraction of the 8th most used pitch class
p_ph09	Fraction of the 9th most used pitch class
p_ph10	Fraction of the 10th most used pitch class
p_ph11	Fraction of the 11th most used pitch class
p_ph12	Fraction of the 12th most used pitch class
p_keySignatureChanges	Number of times the key signature changes
p_accidentals	Percentage of notes that are accidentals outside the key signature
p_mostCommonPitchPresence	Fraction of notes that correspond to the most common absolute pitch
p_strengthOfTopPitches	Second most common pitch by most common pitch. The higher the number the less difference in magnitude between the first two

Table 3.4: Pitch attributes

Melody features

Melody is a sequence of notes that usually change in rhythm and/or pitch. In the rhythm and pitch feature sections, global statistics were extracted. In the melody features, the progression of the different rhythms and especially pitches during the score will be considered. The most important component to analyse will be the interval. The melodic interval is the distance in steps or semitones between a note and the adjacent. For example between the note C and E there are 4 steps, and the interval is called Major 3rd. The list of all the simple intervals and the number of steps are shown in Figure 3.5

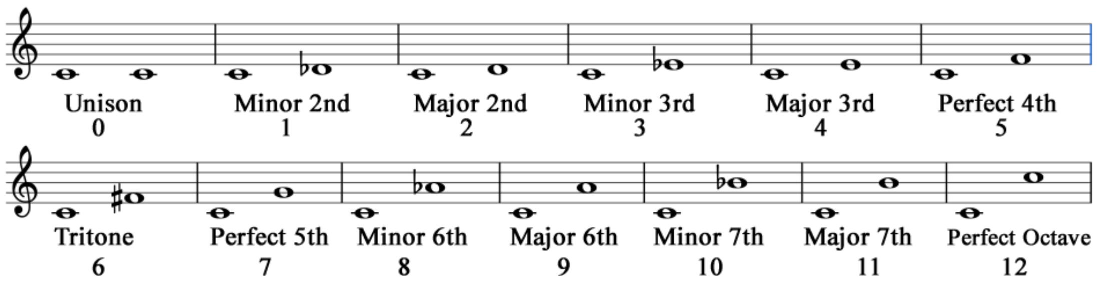


Figure 3.5: Melodic simple intervals. Common name and number of steps below

When the interval is larger than 12 steps (an octave), it is called a compound interval and for analysis purposes, it is usually reduced to a simple interval. It is done just by moving both notes to the same octave. For example C3 and E4 have 16 steps and it is a Major 10th but if they are in the same octave (C3, E3), then there are 4 steps, Major 3rd.

The dataset of scores used in this thesis is for piano which is a polyphonic instrument. In polyphonic music, more than one note can sound at the same time, as opposed to monophonic, where there is a single melody. These melodies have a pattern and meaning in the score and they are all important. In order to create a list of all the intervals found, the melodies were first extracted. Melody extraction in polyphonic music is a field in MIR research that has many challenges, as many melodies are involved at the same time (Corrêa & Rodrigues, 2016; Isikhan & Ozcan, 2008). To simplify the process to convert polyphonic music to multiple monophonic sequence notes the following was considered:

- The notes in each part/staff were evaluated as different melodies.

- If a part had more than one voice, each voice was handled as a melody too.

- If there are still melodies with more than one note played at the same time, only the one with the highest pitch would be considered.

Once all the melody lines are extracted from the score, a list of all the intervals between a note and the next is built and features can be created from it, such as interval histograms, melody direction (if the pitch goes up or down), average and standard deviation, unique number of different intervals, or if the melody has a trend to go to higher or lower pitch, or the amount of changes in duration along the song.

Table 3.5 details all the melody attributes extracted from each score.

Attribute	**Description**
Histogram of melodic intervals (between consecutive notes)reduced to simple intervals (same octave)	
m_intervals_01_minorSecond	Fraction of intervals that are minor second
m_intervals_02_majorSecond	Fraction of intervals that are major second
m_intervals_03_minorThird	Fraction of intervals that are minor third
m_intervals_04_majorThird	Fraction of intervals that are major third
m_intervals_05_perfectFourth	Fraction of intervals that are perfect fourth
m_intervals_06_Tritone	Fraction of intervals that are tritone / augmented fourth
m_intervals_07_perfectFifth	Fraction of intervals that are perfect fifth
m_intervals_08_minorSixth	Fraction of intervals that are minor sixth
m_intervals_09_majorSixth	Fraction of intervals that are major sixth
m_intervals_10_minorSeventh	Fraction of intervals that are minor seventh
m_intervals_11_majorSeventh	Fraction of intervals that are major seventh
m_intervals_12_Octave	Fraction of intervals that are octave
Direction of the melody	
m_directionUp	Fraction of melodic intervals that goes to a higher pitch
m_directionDown	Fraction of melodic intervals that goes to a lower pitch
m_directionSame	Fraction of melodic intervals that keeps the same pitch
Melodic interval statistics	
m_mostCommonMelodicInterval	In semitones, the most common melodic interval found
m_mostCommonIntervalPresence	Fraction of use of the most common melodic interval found
m_melodicIntervalAverage	Average in semitones of the melodic intervals (within an octave)
m_melodicIntervalStd	Standard deviation in semitones of the melodic intervals (within an octave)
m_melodicDistanceAverage	Average in semitones of the melodic intervals (absolute distance)
m_melodicDistanceStd	Standard deviation in semitones of the melodic intervals (absolute distance)
m_numberOfMelodicDistances	Number of unique melodic distances (with a 5% minimum presence).
m_largeIntervals	Fraction of intervals that are greater than an octave (12 semitones)
m_melodicDurationChange	Fraction of notes that changes duration in the melody. For example, if one note is half duration and next is other duration different than half, then there is a rhythm change
m_melodicContourChange	Fraction of intervals that don't follow the same direction than the previous one. For example, if the previous interval was going up and the next is also going up, there is no contour change.

Table 3.5: Melodic attributes

Harmony features

Harmony is related to the simultaneous combination of notes. These notes played together, form chords that can be analysed using the distance in pitches between the ones in that chord. All the vertical intervals between all pairs of notes in a chord will be gathered from all the chords found in the score. The difference with the melodic features, is that in this case only the vertical relationship between notes is considered, as opposed to the horizontal relationship, from left to right of the score.

Harmonic intervals histogram can be created using the distance between notes. There will be twelve bins that will contain the percentage of use of each interval, the same way it was handled in the melody features section, as seen in Figure 3.5.

By analysing all the notes in a chord as a single unity, the chord type can be determined. Chords with three pitches are called triads and the most typical ones are minor, major, diminished or augmented. Other types of chords with more than three pitches are also considered. In Western music, chords are built with intervals of a third stacked over the root note. Figure 3.6 shows the main chord types based on the C note. In order to find out the type when the note is other than C, the same relationships between the pitches of notes is used. For example, in the Major triad (notes C, E and G), the intervals are a Major third (4 steps) plus a Minor third (3 steps), exactly the same relation when using the notes G, B and D. There are many chord formation types but only the most common are considered as individual features, the rest will be grouped in features such as *h_chordType_09_Other4Chords*, that include the percentage of use of chords with four notes that are not major seventh or minor seventh or dominant seventh.

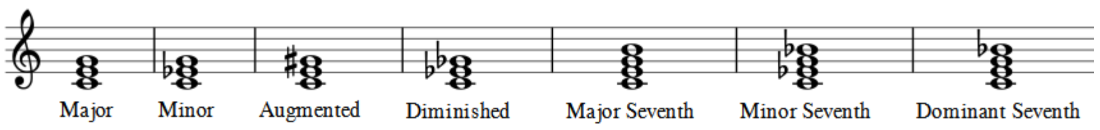


Figure 3.6: Most common chord types, based on the note C

Description of all the harmony features can be found in Table 3.6.

Attribute	**Description**
Percentage of harmonic intervals between pair of notes in all the chords	
h_harmonic_interval_01_m2	Fraction of harmonic/vertical intervals that are minor second
h_harmonic_interval_02_M2	Fraction of harmonic/vertical intervals that are major second
h_harmonic_interval_03_m3	Fraction of harmonic/vertical intervals that are minor third
h_harmonic_interval_04_M3	Fraction of harmonic/vertical intervals that are major third
h_harmonic_interval_05_P4	Fraction of harmonic/vertical intervals that are perfect fourth
h_harmonic_interval_06_TT	Fraction of harmonic/vertical intervals that are a tritone
h_harmonic_interval_07_P5	Fraction of harmonic/vertical intervals that are perfect fifth
h_harmonic_interval_08_m6	Fraction of harmonic/vertical intervals that are minor sixth
h_harmonic_interval_09_M6	Fraction of harmonic/vertical intervals that are major sixth
h_harmonic_interval_10_m7	Fraction of harmonic/vertical intervals that are minor seventh
h_harmonic_interval_11_M7	Fraction of harmonic/vertical intervals that are major seventh
h_harmonic_interval_12_P8	Fraction of harmonic/vertical intervals that are perfect octave
h_topVerticalInterval	Vertical interval most used as string. (m2…. P8)
h_topVerticalIntervalPresence	Fraction of the vertical interval most used.
h_partialChords	Fraction of chords that have only two notes by the duration of the score in units.
Chord features	
h_chordNotesAverage	Average number of notes per chord in the score
h_chordNotesStd	Standard Deviation of number of notes per chord in the score
h_chordPitchClassesAverage	Average number of different pitch classes per chord
h_chordPitchClassesStd	Standard deviation of the different pitch classes per chord
h_chordsWithOctave	Fraction of chords that include the same pitch class more than once
h_uniqueChords	Ratio of combinations of chords found. Unique chords by total number of chords used.
h_uniqueChordDurations	Number of distinct chord durations
h_chordDurationAverage	Average duration of the chords
h_chordDensity	Average of chords per score duration unit (quarter length)
h_mostCommonChordSet	Percentage of use the most common chord (pitch class set)
h_chordType_01_minorTriads	Fraction of chords that are minor triads
h_chordType_02_majorTriads	Fraction of chords that are major triads
h_chordType_03_diminishedTriads	Fraction of chords that are diminished triads
h_chordType_04_augmentedTriads	Fraction of chords that are augmented triads
h_chordType_05_OtherTriads	Fraction of chords that are triads but not in any of the above groups
h_chordType_06_MinorSeventh	Fraction of chords that are minor sevenths
h_chordType_07_DominantSeventh	Fraction of chords that are dominant sevenths
h_chordType_08_MajorSeventh	Fraction of chords that are major sevenths
h_chordType_09_Other4Chords	Fraction of chords that contains four notes but not in any of the groups
h_chordType_10_ComplexChords	Fraction of chords that have more than four different pitches

Table 3.6: Harmony attributes

3.4.2 Data cleaning

Data quality assessment is an important step for any classification task or in general, any data mining system. Issues have to be identified because data with low quality will not give the appropriate results. This assessment starts by exploring the data to point out the different issues, such as missing data, outliers and inconsistencies. It will follow by choosing the approach to take in order to fix or at least minimise the impact of the problems.

Once all the features have been extracted, a review analysis of the feature values must be performed. Descriptive statistics will be generated about each of the features. Depending if the data type from the feature is continuous or categorical, different measure values have to be analysed. For the features that are numeric the count, minimum, maximum, mean, standard deviation, 1st quartile, 3rd quartile and median should be reviewed. Minimum and maximum values give an idea of the ranges. Mean and standard deviation help to understand the variation and central tendency of the features. Histograms of all the continuous features can also be included for the data quality assessment to visually check the distribution of values. For categorical values, the cardinality, mode and mode% will tell us about how many levels are included, the most common value and its frequency.

Analysis of this information can help to identify data quality issues the most common being missing values and outliers (Kelleher, Namee, & D'Arcy, 2015)

Missing values

The number of records with missing values for each of the features can be easily identified using the previous summary information. The cause for having missing values has to be identified before proceeding with a solution. It may be due to an error when creating the table with the features or that there was not enough information for that particular attribute in that record.

For this experiment, as the feature values were calculated from the music scores, the only possible reason for the presence of missing values would be an error in the feature extraction process. The feature extraction process would then be fixed accordingly. In the case that, for a particular experiment, the researcher does not have access to the raw data source and control over the construction of the features, other methods have to be used to deal with the missing values (Han & Kamber, 2011). The common approaches are:

- *Delete the observation/row*: This is an extreme method to use because by ignoring this instance from the model, other important attributes would be ignored too. Unless there are many features in that record with missing values, this method leads to an unnecessary loss of data information.

- *Impute the value*: replace the missing value with the most reasonable value. The most common values to use are the mean, median or even calculated from a tendency using a predictive model such as linear regression.

Outliers

An outlier is a value that deviates substantially from the mean. As in the case of missing values, it has to be identified if the outliers are invalid or valid. Invalid outliers could be present due to an error in the feature extraction process and, in this particular experiment, it could also be an error in the source file data (music score file). Valid outliers are values that correctly represent the information contained in the dataset but just happen to be really different from other values.

In order to check the presence of outliers, minimum and maximum values for each feature can be analysed using the domain knowledge and the description of the features. For example, all the features representing a fraction or percentage should have their values between 0 and 1, or a feature that indicates the number of pitch classes used in the score cannot have a value higher than 12, as there are no more than 12 pitch classes. The use of boxplots or checking the gap between 3rd quartile and the maximum value, are other ways that can be used to find outliers.

If the outliers are valid values, there are methods to minimise the impact of these in the models without deleting the observations. These methods includes data transformation techniques, such as *binning* or *clamp* transformation.

Clamp transformation is an easy way to deal with outliers. With this transformation, values higher and lower than a specified threshold, will be replaced by the threshold values. The thresholds can be set based on domain knowledge of the feature or a calculation using the quartile values from the summary statistics.

3.4.3 Data transformation

Data transformation techniques are used to consolidate the data to a more suitable form to be used to train the models or for other statistical analysis. A transformation can also be used to deal with inconsistency and outlier problems. There are different approaches to transform the data such as binning, clamp transformation, derived features and normalisation.

Binning

Binning is a data reduction technique used for numeric attributes. It consists of converting a continuous feature into a categorical feature. An example would be transforming a field representing the age into age groups. Each group is a bin that contains all the values from a defined range. The bins could have the same number of records in each (equal-frequency binning), or the same range of values (equal-width binning). There are advantages of using binning, including handling outliers, as the outliers will be included in the first or last bin, along with other values (Kelleher et al., 2015).

Handling categorical features

Most classification machine learning algorithm implementations present better results if using only numeric features, and in some libraries, like the one used in this experiment, the scikit-learn python package, all the variables have to be numeric.

The easiest way is to convert a single categorical feature into different binary features. The number of the new dimensions will be the cardinality or number of unique values of the categorical feature. If for example, we had a categorical variable with three different values, A, B or C, three new binary variables will be created. Each of the variables will account for the presence or absence of that value in the feature space. After that, the original categorical feature can be removed from the feature space.

If the categorical feature represents an interval that can be ordered, then it could be converted into a numeric feature. For example, a categorical feature that represents a client income could have the values ‘low’, ‘medium’ and ‘high’ and they can be transformed to 1, 2 and 3 and be used as a number.

Additionally, a numeric feature could actually hold a categorical feature if the values do not represent a quantity, an interval or ratio. In other words, all the different values are independent to each other. In this case, the feature should be treated as categorical.

Derived features

New features can be constructed based on the features that are already present on the dataset. Two or three features that represent the same main concept might be aggregated in a new attribute that could simplify the dataset and at the same time, improve the predictive model (Han & Kamber, 2011). On the other hand, flag features that contain a Boolean data type can be created to represent the absence or presence of a property, using one or a combination of feature values to calculate the flag value.

Normalisation

The numeric features used in any classification task, normally have different measurement units or ranges. Depending on the machine learning algorithm, the model predictions could be affected by using these values directly. For example, a nearest neighbour algorithm would find the observation that is a closest match based on the distance that is calculated using the values. Therefore, the feature that has a greater range would have more impact in the calculations than another feature with a shorter range.

Normalisation can be used to change that numeric feature to fall in a range but still keeping the relative distance between their values. This approach has to be applied to all the numeric features that will be used to create the model in order to standardize the scale. It can be applied by using a min-max normalisation using a specific range usually from 0 to 1, or using the Z-score standardization, that is widely used in statistical analysis and uses the mean and standard deviation to calculate the new values.

Min – Max normalisation

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

Z-score standardization

$$Z_i = \frac{X_i - mean(X)}{sd(X)}$$

3.5 Modelling

3.5.1 Group features in categories

In order to compare the effectiveness of each of the categories of features, a classifier model will be created with each group selection. A final classifier using the whole set of features will be also built and it is expected that the combination of all the categories would give the best results. Table 3.7 shows the predictive models that will be created with the final number of features included in each.

Classifier	Category	Number of features
1	Rhythm	32
2	Pitch	25
3	Melody	25
4	Harmony	35
5	All features	117

Table 3.7: List of classifier models built with the number of features

A unique classifier with exactly the same parameters will be used to compare the different models created. This is an important step as the change in any parameter before the creation of the next model would make the comparison unreliable. These models can be further compared and evaluated with no bias of the machine learning algorithm.

3.5.2 Variable ranking

Another research objective of the study was to find out what specific music attributes were the most important contributors to identify the musical period of a music score.

The machine learning algorithm that will be used to create the models is the SVM. This MLA often gives good results but it is considered a “black box” algorithm and the results are hard to interpret, especially if using a non-linear kernel method (Ben-Hur & Weston, 2010). The output doesn’t provide information about the importance of the features used to create the model.

By identifying the most important features, new predictive models could be created discarding the irrelevant attributes. These new models could improve the accuracy, reduce training time and reduce the risk of over-fitting (Guyon et al., 2003). The models created would be easier to understand and improve generalisation. In many cases, machine learning algorithms such as SVM or neural network in the sklearn Python package, perform a feature selection process internally before creating the models.

For this experiment, the Random Forests algorithm will be used as the researcher has access to the information of feature importance and it is a robust and popular algorithm

(Genuer et al., 2011). The random forest is an algorithm that creates multiple decision trees, each with a different random subset of the available data (Breiman, 2001). The classification results are given by the average prediction of each tree. The importance of each feature is calculated by aggregating the importance of that feature in the individual trees.

Knowing what individual features are the most important predictors will give insights related to music theory and the music style of the different periods.

3.5.3 Support vector machine classifier

As mentioned in the literature review, it was decided to use the support vector machine algorithm to create the models to compare the set of features. The literature shows that this classifier has been applied in symbolic genre classification with success (Corrêa & Rodrigues, 2016; Xu et al., 2003). Support vector machine (SVM) is a supervised learning model that can be used for regression and classification tasks. The SVM models are well suited to avoid over-fitting and they have a good performance with high-dimensional datasets (Kelleher et al., 2015).

The support vector machine algorithm is based in the principle that it is possible to separate the instances of different classes with an optimal linear boundary. In cases with more than two dependant features, that line would be a hyperplane. That linear decision boundary separates the classes with a ***margin,*** that is the distance to the closest observation or training instance. A linear decision function is used to find the largest optimal margin between the vectors of the different classes. A larger margin to divide the classes would be more accurate (Cortes & Vapnik, 1995). If the data cannot be separated by a linear hyperplane, **kernel** methods can be used by mapping the data on to a high-dimensional feature set that can be divided linearly. A decision has to be made on what kernel method to use (linear, radial or polynomial), and the SVM parameter settings. The most important parameters are the soft-margin (C) and the gamma value. The C value defines the margin width to separate the classes and is a trade-off between penalty for misclassification and complexity of the model (Cortes & Vapnik, 1995). A small value of C leads to a large margin and a large C value leads to a small margin. The gamma value is the kernel coefficient if using a radial kernel. Low values of gamma will lead to a decision boundary that is almost linear (similar to using a linear kernel). High values of gamma will lead to a greater curvature of the decision boundary to adapt to the training set (Cortes & Vapnik, 1995). A value too large could lead to over-fitting, as values in the training set could represent unusual information and the model is not flexible enough to generalise in the test set (Ben-Hur & Weston, 2010)

3.6 Evaluation

The performance of each category of features (rhythm, pitch, melody and harmony) will be evaluated individually. **K-fold cross validation** will be used to test the performance of the models. With the results obtained, the **accuracy** will be calculated to determine the best model to classify music scores. C**onfusion matrices** will be created for each category to analyse the accuracy and misclassifications in each class. Different metrics will be calculated using the confusion matrix in order to further evaluate the models and examine how the predictors behave in each musical period. A whole list of all the music pieces with their predictions will also be gathered to be analysed at a composer level.

In order to determine what individual attributes inside each category are the most important to identify the musical period, box plots and random forest tree algorithm will be used.

3.6.1 K-fold cross validation

K-fold cross validation will be used to test the performance of the models. With a K-fold validation, the dataset is divided into equally sized K folds. One of the folds is used for testing and the rest are used for training the model. This is performed K times, each time with a different fold as testing. Using this method, all the instances will be used exactly once for testing and k-1 times for training. All the results are added up and averaged if needed. A stratified method of sampling was used to maintain the same proportion of instances per class in each fold. An extreme case of K-fold is when k is equal to the number of samples, this is called leave-one-out, and only one sample is left for testing in each fold. This method is computationally expensive and it is normally used with small datasets when there is not enough data to dedicate for training the model (Kohavi, 1995)

Ten folds will be used as this is the most common cross-validation method used and it is the number recommended for estimating accuracy due to the better bias and variance trade-off compared to other methods (Cawley & Talbot, 2010).

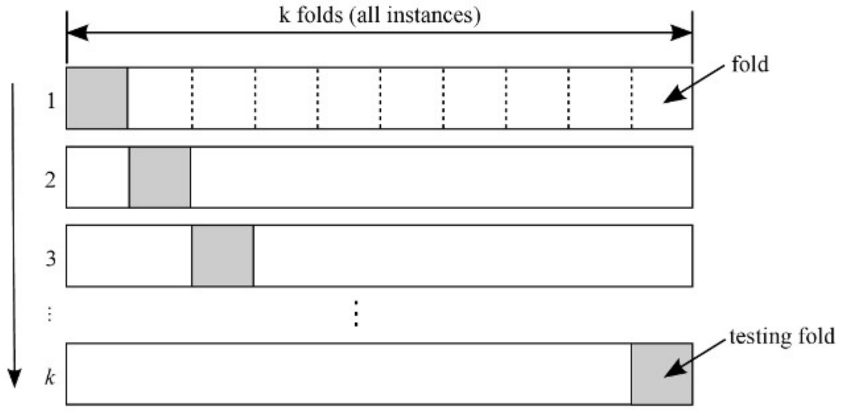


Figure 3.7: K-fold cross validation method
(Borovicka, Jr., Kordik, & Jirina, 2012)

3.6.2 Confusion matrix

A confusion matrix gives more details of the evaluation and it is used to calculate other performance metrics. In this matrix the predictions made by a model in each of the classes is shown. For a binary classification there will be four outcomes with two different levels (usually referred as positive or negative). The structure of a confusion matrix is shown in Table 3.8

		Predicted	
		Yes	**No**
Actual	**Yes**	TP	FN
	No	FP	TN

Table 3.8: Confusion matrix for a binary classification

Each cell shows the number of times a different outcome was predicted in the test set. Based on the table presented before, there are four values:

True positive (TP): number of times the Yes class was predicted as Yes

True negative (TN): number of times the No class was predicted as No

False positive (FP): number of times the No class was predicted as Yes

False negative (FN): number of times the Yes class was predicted as No

A confusion matrix can be extended to represent the outcome of a multiclass classification problem. For the purpose of this dissertation, a cross tabulation with the music periods and the model predictions was created for each of the feature sets.

		Predicted			
		Baroque	**Classical**	**Romantic**	**Modern**
Actual	**Baroque**				
	Classical				
	Romantic				
	Modern				

Table 3.9: Structure of a confusion matrix for the musical period classification

With this table we can evaluate the accuracy and misclassifications in each period.

3.6.3 Evaluation metrics

Accuracy is an overall measure that gives an estimate of the performance of a model. It is the proportion of correct classifications by the total number of instances.

$$Accuracy = \frac{\text{number of correct predictions}}{total\ predictions}$$

When using a confusion matrix, the correct classifications are in the diagonal values. The sum of all these values divided by the number of instances would give the accuracy. Although accuracy is the most extended measure to evaluate performance, it can sometimes hide other problems in the model. For example, in a binary classification when the target class is not balanced, a high accuracy doesn't mean good quality predictions. If, for instance, a dataset has 90% of records from a class, a dummy model could just give all the predictions to that class and the accuracy would be of 90%. In this case none of the records from the other class would have been correctly classified.

Other metrics such as **recall** and **precision** are needed to further evaluate the model.

$$Recall = \frac{\text{TP}}{TP + FN} \qquad\qquad Precision = \frac{\text{TP}}{TP + FP}$$

Recall measures how effective a classifier is to identify all the instances from a class. **Precision** tell us how confident the model is that when it predicts an instance to have a particular class, it is correct.

In the following Table 3.10, there is an example of a confusion matrix with the musical period as target classes and with recall and precision calculated.

		Predicted				
		Baroque	Classical	Romantic	Modern	**Recall**
Actual	Baroque	6	1	1	0	0.75
	Classical	0	9	0	1	0.90
	Romantic	0	5	8	3	0.50
	Modern	0	2	2	1	0.20
	Precision	1.00	0.53	0.73	0.2	

Table 3.10: Example of recall and precision metrics

In the above example, Baroque has a recall value of $(\frac{6}{6+1+1+0}) = 0.75$ and a precision value of $(\frac{6}{6+0+0+0}) = 1$. This indicates that for the *Baroque* period, the model had a 75% recall. The dataset had 8 baroque pieces and 6 of them were correctly classified. On the other hand, this model has a 100% precision for *Baroque* class, meaning that every time that the model predicted a song as baroque, it was correct. A simple way to calculate the global precision of the model is by averaging the class precisions. There are other methods that give different weights to each precision depending on the class.

Accuracy was used as the overall measure to determine what group of musical features (rhythm, pitch, harmony and melody) performs the best for music period classification. Other metrics such as recall and precision will be calculated for each feature category and class, and they will be discussed.

Finally, although the scope of the study was the classification by musical periods, the predictions made at a composer level will be evaluated too.

3.7 Conclusion

This chapter included the details of the design that will be followed to perform the experiments in this research. Keeping in mind that the goal of the research is to analyse which feature categories have more of an impact on determination of the musical period of classical music score, a design and methodology was created to achieve this goal. Music scores will be collected and prepared, and target labels will be assigned. Then the features will be extracted from those files and data exploration analysis will uncover issues in the feature extractor just created or in the source data files. This exploration of data will allow preparation of the data and performance of the data transformations that can help optimize the dataset to be used for the models. Once the models are created, they will be evaluated using different metrics. Feature selection techniques will provide insights about the most important attributes in this classification.

One of the **strengths** of the design is that the researcher has access to a primary source of information, giving scope for feature extraction and engineering based on the researcher´s domain knowledge of music theory. Although the features proposed in this study are reasonably extensive, more features could be added by updating the feature extraction process.

Limitations of this design includes the size of the dataset, around 850 music files, which might not cover all the different characteristics from all the periods. An effort was made to obtain the most significant works from each composer and period for piano but it was limited to the available data.

In the next chapter, details of the implementation to perform the experiment according to this design and methodology will be described.

4. IMPLEMENTATION AND RESULTS

4.1 Introduction

This chapter will outline the details of the implementation. It will start with a description of the implementation details to collect the music scores to build the dataset. The assignment of labels to each of the records will be discussed in detail. Then the construction of the features from the raw files will be described. Data preparation details are discussed and the first findings shown. Finally, implementation of the models and evaluation of the results will be detailed.

4.2 Data Collection

4.2.1 Music files

Music files from the KernScores library were selected browsing the web. Only files from the piano instrument were considered and only files from the four musical periods: Baroque, Classical, Romantic and Modern were selected, as mentioned in the design section 3.3.2. Zip packages were downloaded and extracted directly to a local folder. Each file was a humdrum format text file containing all the information for a single score.

MusicXML files were also downloaded, to complement the score files already collected, especially scores from the Modern period. These files were downloaded mainly from musecore.com. Python scripts were implemented and run in order to automate the collection of the files, following this process:

1. A python script was run to crawl the website and extract information from each score web page found. The information was exported in an excel file, including: composer, title, instruments and download link, for each score. The *BeautifulSoup* python library was used to help create the crawling script.
2. Using filters, a selection of the scores was made based on the instrument (piano) and avoiding duplicates. This selection was flagged in a new column.
3. A second script was run to download the scores using the URL extracted in the step 1, with the selection of scores made in step 2.

4.2.2 Labels

As mentioned in section 3.3.3, all the records of the dataset needed a class representing the musical period in order to perform the classification tasks. This target class was not present in the music files and had to be added based on the composer.

In order to assign the class to each of the instances in the most efficient way, the use of a reference table was needed. A table including composer name and period was acquired from the internet (https://www.mfiles.co.uk/composer-timelines-classical-periods.htm) and the following process was followed:

- The composer name was extracted from the metadata of each of the music files.
- A python script was developed and used to loop through all the records of the dataset, perform a lookup operation on the composers reference table and get the closest match using the Jaro-Winkler distance for string similarity.
- The script would then obtain the musical period of the closest match found in the reference table and add it to the dataset. The composer would be also added to the list.
- Lastly, a manual check was performed on the final dataset to fix minor issues including:
    - *Missing composer (no class assigned):* the records that did not have the composer included in the metadata of the music file had to be manually added, the information was taken from the source where it was obtained. The number of records with this issue was very small.
    - *Incorrect composer (wrong class assigned)*: The composer was compared with the closest match to check if that was the correct name. This issue again was very minor as using Jaro-Winkler distance, the correct name was found in most of the cases. For example, 'Debussy', 'C. Debussy', 'Claude-Achille Debussy' or even 'Claude Debussy (1862 - 1918)' names, all matched with the composer 'Claude Debussy', and the period 'Modern' was assigned correctly.

4.3 Data preparation

Following the design of the experiment, the next step was the data preparation. This started with the construction of the features from the raw files. After that, the feature values were reviewed with descriptive statistics to identify quality issues that were then fixed. Finally, some data transformations were applied to the dataset to prepare it before modelling.

summary statistics of the independent variables were created, as well as histograms and boxplots. This information helped to understand the data being used and evaluate the value distribution of the features.

4.3.1 Feature extraction

At this stage the information available in the dataset, after the scores were downloaded and labelled, was: composer, title, period and filename path. An example of the information stored so far, can be seen in Table 4.1. Composer and title were left in the dataset in case further investigation about the predictions needed to be carried out.

id	**composer**	**title**	**period**	**filename_path**
1	Claude Debussy	1st Arabesque	Modern	/musescore/1st_Arabesque.mxl
2	Frederic Chopin	Mazurka 52 in C Major	Romantic	/kern/chopin/mazurka/mazurka-52.krn
3	Domenico Scarlatti	Sonata in G Major,L.333	Baroque	/kern/scarlatti/L333K425.krn
4	W.A. Mozart	Divertimento B-flat Major	Classical	/kern/mozart/piano/sonatina/k439b.krn
5	Erik Satie	Gymnopedie 1	Modern	/musescore/Satie_Gymnopedie1.mxl
6	J. S. Bach	Well-Tempered Clavier 2, Fugue n1	Baroque	/kern/bach/wtc/wtc1f01.krn
-	-	-	-	-
-	-	-	-	-

Table 4.1: Example of initial dataset before feature extraction

Using the path location of the file, all the music scores were opened and the features created from the information contained in the files. In order to parse the files to access the notes pitch and value information, the music21 python library (Cuthbert & Ariza, 2010) was used. The Music21 is an open source python package for computer-aided musicology with a set of tools to perform operations with symbolic music. For example, the researcher can gather information about a score for music analysis, or create a new one using complex computer music generation algorithms.

For this experiment, music21 was used to process the music score downloaded in order to obtain the information needed to create the features. The advantage of using this particular package is that it was not restricted to a unique format file and both humdrum and MusicXml files could be opened. Once the file had been parsed, a python object was accessible with all the information of the score. An example of how the information was extracted and prepared to be analysed can be seen in the next script:

```
s = ms.converter.parseFile(filename)
for note in s.flat.notes:
        # access information about each note
        duration = note.duration
        pitchClass = note.pitch.pitchClass
        pitchNumber = note.pitch.ps
        score_notes_info.append((duration, pitchClass, pitchNumber))


dfScoreNotes = pd.DataFrame(score_notes_info, columns=['duration','pitchClass', 'pitchNumber']
```

The previous script loops through all the notes of the score to extract basic information such as note duration, absolute pitch (pitch number) and pitch class and is stored in a pandas dataFrame. Using this representation of the data from a score as a data subset, global features can be calculated, such as duration range, minimum or maximum pitch or duration, number of different pitches used or most common note duration, as shown in the next example:

```
s = ms.converter.parseFile(filename)
df = createNotesDataFrame(s)

info['r_minimumDuration']: df['duration'].min()
info['r_maximumDuration']: df['duration'].max()
info['r_noteDurationAverage']: df['duration'].mean()
info['r_noteDensity']: len(df) / float(s.duration)

dfDurationTotals = df['duration'].value_counts(normalize=True)
info['r_mostCommonNoteDuration']: dfCounts.index[0]
info['r_mostCommonNotePresence']: dfCounts.iloc[0]
info['r_numberOfDistinctDurations']: len(dfCounts)
```

Apart from a list or dataframe with information about each note in the score, other datasets from a single score were created to calculate other features:

- A list containing all the intervals or distances between one note and the next was used to create most of the melodic features.
- A list with all the occurrences of three or more notes played at the same time was used to gather the chord type information for the harmonic features.
- A list of all the distances between the notes that are played together was used to create the harmonic intervals

Other more specific features were created by accessing other types of information in the score. For example, the time signature and key signature values for each measure, allow us to know how many times they change along the score.

To build all the features, this operation was performed on each of the music files, and the data extracted added to the original dataset, along with the music period (target class).

The content of real music scores could range from a few hundred to thousands of notes with different durations, pitches and in different parts. The complexity of this data source made it difficult to check and assess the quality of the information extracted. In order to create the features that were described in the design section of this dissertation, a set of short music score test files was created. These test files can be seen in Appendix C. The feature extraction script was implemented and then applied to these test files. These test files could be opened with a music notation software to manually check that the feature values were correct.

Once the method used to extract the features was optimised, it was executed on the music score dataset that was downloaded, to obtain a dataset that could be used to perform analysis and classification operations. Summary statistics were generated to help preparing the data before creating the models.

4.3.2 Data cleaning

Summary statistics of the independent variables were created (as seen in Appendix B), as well as histograms and boxplots. This information helped to understand the data being used and evaluate the value distribution of the features.

After analysing the descriptive statistics, **missing values** were found in the chord type features from the harmony category. These represented only 5% of the scores and once the process was reviewed, it was found that for those scores those missing values were correct. They showed that 0% of that chord type was found in that score. Therefore, this was fixed in the feature extraction step by using a zero value instead.

The **outliers** were identified following the data exploration described in chapter 3. There was a small number of outliers that turned to be **invalid** values due to an error in the implementation of the feature extraction process or in the music score file. This latter error was found in no more than a handful of scores which were then manually fixed by editing the file with a notation software. Errors in the implementation were corrected and the feature extraction scripts run again.

In the majority of cases, the outliers were **valid** values extracted from the scores. Some examples of features that have outliers can be seen in Figure 4.1. The attribute that represents the number of different notes used in a song (p_numberOfPitches) has an average of 46.85. A look at the 1st and 3rd quartile values shows that half of the instances are in the range of 39 and 55, indicated by the dark area in the boxplot diagram. Values above the upper whisker or below the lower whisker are considered outliers. In this case, music scores that use more than 78 different notes or less than 17, are outliers.

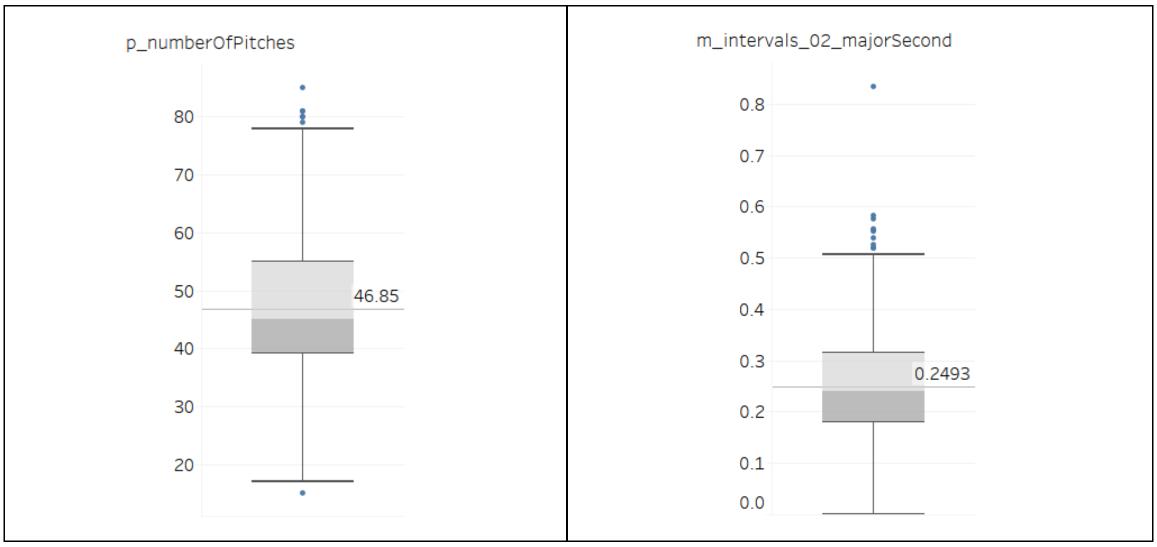


Figure 4.1: Example of features with outliers

Records with these outliers were not excluded from the model for a number of reasons. The dataset was not big enough to ignore all the rows with outliers, resulting in a smaller dataset that would not be suitable for the experiment. On the other hand, outliers might contain important information about the data in combination with other features, or in some cases the outliers could have a pattern and come from the same target class. In the example above, there were 5 outliers in the feature p_numerOfPitches, from which 3 were from the ‘Romantic’ class and the other 2, from ‘Modern’.

Therefore, the outliers were kept and to minimise the impact of the outliers on the model, a clamp transformation was applied, as mentioned in section 3.4.3. The threshold values were chosen following the common rule of thumb of using 1.5 times the interquartile range plus the third quartile, and 1.5 times the interquartile range minus the first quartile, for the upper and lower threshold respectively (Han & Kamber, 2011).

4.3.3 Data transformation

As part of preparing the data to train the model, **derived features** were created. Derived features are attributes created with the combination of already existing features. Sometimes the derived feature can replace an existing feature. These new attributes, based on background knowledge, can provide facts that are not represented in the current feature set. The following changes in the feature set were applied.

r_durationRange = *r_maximumDuration – r_minimumDuration*

This represents the range of durations used in a score in unit of the crotchet note and using decimals.

p_pitchRange = *p_maximumPitch – p_minimumPitch*

This represents the range of note pitches used in a score (the highest minus the lowest pitch found)

Other features of the dataset were created after analysing the summary statistics and histograms. The distribution of the features that represent the number of changes in the key or time signature along the score is highly skewed, with most of the values being 0. In general, changes in the key or time signature are not very common in Western music, therefore even just one change is considered an important characteristic. The presence of music scores with many changes, will make the ones with just one change less important than they are. In order to simplify these features, they were transformed into binary types, representing the presence or absence of key or time signature changes. In this case, the original features are not needed and they can be replaced with the new values.

r_timeSignatureChanges = (*r_timeSignatureChanges == 0)*

p_keySignatureChanges = (p*_keySignatureChanges == 0)*

Another step in the data preparation is to **handle categorical features** to be used in the models. As described in design section 3.4.3, most machine learning algorithms or statistical analysis do not handle categorical features and therefore, need to be converted to numeric or binary features. In the dataset of the experiment, the only categorical features are *h_topVerticalInterval* and *m_mostCommonMelodicInterval*, indicating the name of the intervals most used, harmonic and melodic, respectively.

Using the following script, one new binary dimension was created for each different value in the categorical feature. After that, the original feature was removed from the dataset as it was not needed anymore.

```
for i in dfDataSet['h_topVerticalInterval'].unique():
        dfDataSet['h_topVerticalInterval_' + i] = (dfDataSet['h_topVerticalInterval'] == i)
dfDataSet.drop('h_topVerticalInterval', axis=1, inplace=True)
```

There are other features that are likely to be categorical even if they contain numeric values and they should be handled as categorical variables. If these variables are not detected, they could have a negative impact on the results of the model. For example, if a variable had numeric zipcodes, these should be converted to district/area or at least to a string value so the model will not wrongly treat the feature as a continuous value.

A method to identify these features is to review the summary statistics. Numeric features with low cardinality are likely to be categorical. In Table 4.2 below, features with low unique values are shown. The table also includes minimum and maximum values, the mode (most frequent value) and their percentage.

#	attribute	count	cardinality	mode	mode%	min	max
1	r_mostCommonNoteDuration	878	14	0.5	34.17	0.083	3
2	r_numberOfDistinctDurations	878	34	8	11.16	2	43
3	m_numberOfMelodicDistances	878	10	6	22.55	2	11
4	p_numberOfPitchClasses	878	6	12	77.9	7	12
5	h_uniqueChordDurations	841	28	3	17.77	1	53
6	p_distanceStrongestPitchClasses	878	6	5	74.15	1	6

Table 4.2: List of numeric features with low cardinality

Based on the domain knowledge, features that are correctly represented as continuous values can be recognised. The feature *r_mostCommonNoteDuration* has low cardinality but it is a range of durations, from shortest to longest note duration. The features 2 to 5 in the list, contain the number of appearances of different characteristics in the score, such as how many different durations, how many pitches or how many melodic distances are used. The case of the feature *p_distanceStrongestPitchClasses* is different. This feature stores the distance, in semitones, between the two most important or used pitch classes in the score. These values have no meaning when ordered, as they represent different things. For example, a distance of 5 semitones is a perfect interval, distances of 1 and 3 are minor types, distances of 2 and 4 are major, etc… It is better to have this feature as categorical, rather than numeric.

Normalisation

As previously outlined in the design section 3.4.3, a normalisation transformation is an important step in the data preparation. This transformation will locate all the data in the dataset in the same scale to avoid the algorithm giving a different weight to features with higher ranges. A normalisation using a range between 0 and 1 was applied to all the continuous features.

The *preprocessing* module from the *sklearn* python library was used to apply this normalisation in the pandas DataFrame of features, using the following script:

```
transformation = preprocessing.MinMaxScaler(feature_range=(0, 1))
dfDataSet = transformation.fit_transform(dfDataSet)
```

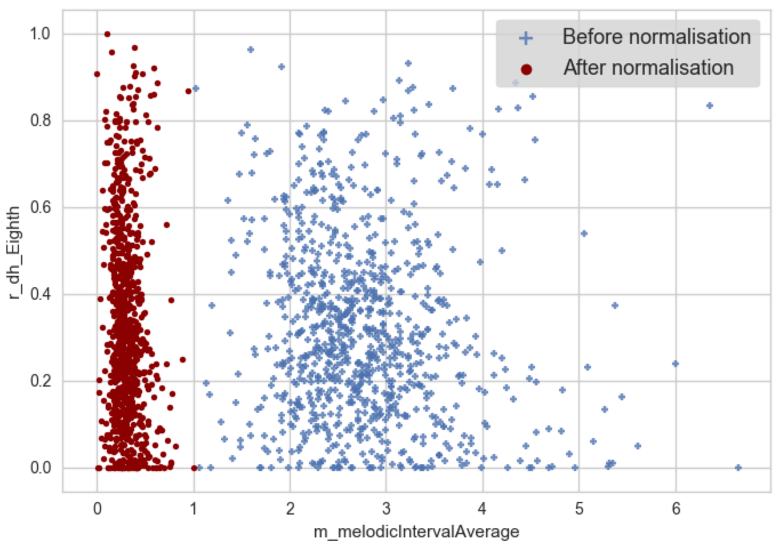


Figure 4.2: Scatter plot of two features before and after normalisation

An example of the effect of applying normalisation can be seen in Figure 4.2. The feature *r_dh_Eighth*, represents the fraction of notes that are Eighth/Quaver notes and ranges from 0 to 1, whereas *m_melodicIntervalAverage* is the mean of the intervallic distances with values from 1 to around 6. After the normalisation, all the values uses the same scale 0 to 1, ensuring there is no bias over the larger range of the second feature. In Figure 4.3, it can be observed in the plot that the transformed features in the new scale (0,1), keep the same relationships they had before the normalisation.

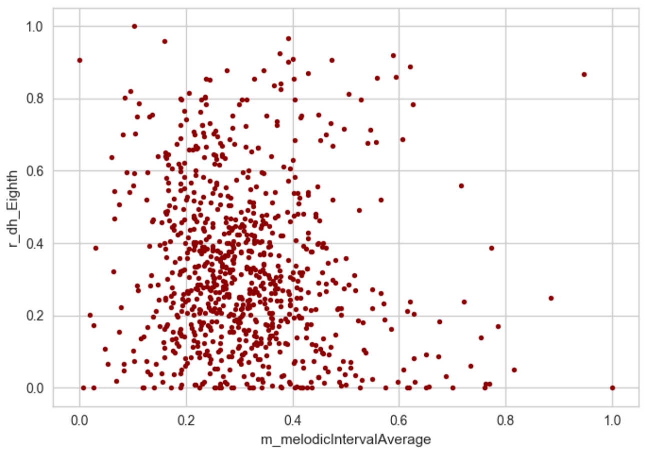


Figure 4.3: Scatter plot of the normalised features in the new scale (0,1)

4.4 Modelling

The python library scikit-learn was used to implement the predictive models and the K-fold cross-validation for this experiment.

The scope of the study was to compare the contribution of different categories of features that are related to musical concepts to predict the musical period of a music piece. For the purposes of the comparison, a model was created for each set of features (rhythm, pitch, harmony and melody). Taking advantage of the naming format of the features names, different datasets with a subset of columns were created. For example, all rhythm features started with ‘r_’, all the pitch features started with ‘p_’ and so on. The original dataset was kept in order to create another model with all the available features. This complete dataset is expected to give the best results. The target set was created by extracting the musical period column from the main dataset.

The division of the datasets in groups of feature sets will allow the experiment to: identify the most important features in each group, and finally create the classification models.

4.4.1 Variable ranking

The main research objective of this dissertation was to determine how the global aspects of a musical composition (rhythm, pitch, harmony and melody), would perform to identify the classical musical period. However, it was also part of the study to try to find out what specific music attributes such as chord types, melodic intervals, pitches, note durations, were more different along the time periods.

In order to investigate the individual contribution of the features in the classification of music scores by time period, a model-based ranking using Random Forest Tree algorithm was used. As mentioned in section 3.5.2, this algorithm, as well as other bagged methods such as Decision Trees, is widely used for variable selection and interpretation of the models by providing a feature importance ranking.

In this experiment, 1000 thousand trees were created with a maximum depth in each tree of 40 levels. The model was fitted for each of the feature categories and a list of the features with its importance was returned.

```
model = RandomForestClassifier(max_depth=40, criterion='gini', n_estimators=1000,
                               oob_score=True, random_state=0)
model.fit(X, y)
featureImportances = model.feature_importances_
dfFeat = pd.DataFrame({'Variable': X.columns, 'Importance': featureImportances})
```

The first five features in the ranking of importance for each of the groups are shown Figure 4.4 below.

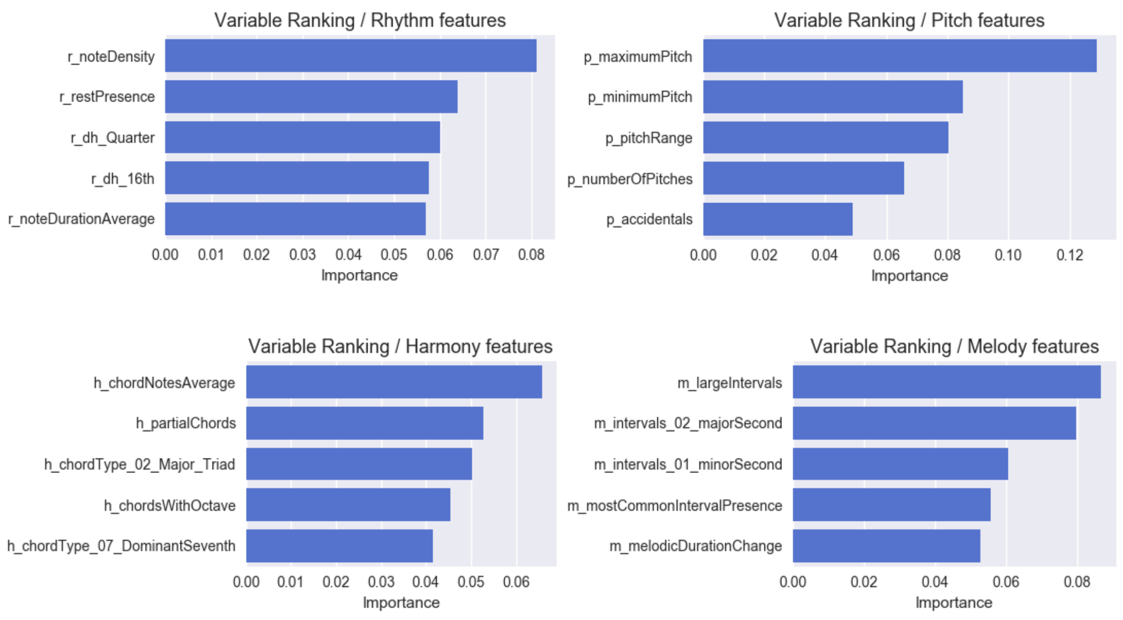


Figure 4.4:Variable ranking with the top five attributes for each feature group

Analysing the contribution of each attribute in the prediction of each class becomes a difficult task, more difficult than in a binary problem. Statistical tests, such as ANOVA, can measure how significant is the difference of the feature value averages for each class. However, in a data mining process, some attributes might be important only if used in combination with other attributes.

For the purposes of an overall assessment of the impact of these features in each period their value distribution with the use of boxplots was reviewed. In order to be able to the understand better the values, the original units and scales were kept, instead of using the normalised values.

Rhythm

The distributions of values per period for the top five features are found in Figure 4.5. The use of quarter/crotchet and 16th/semiquaver notes appears to have changed in the different periods. In the Baroque period there was less use of quarter notes, 16th notes were more commonly used. The opposite is found in the Romantic period. The note duration average attribute shows a similar thing, where shorter note symbols were used in the earlier periods.

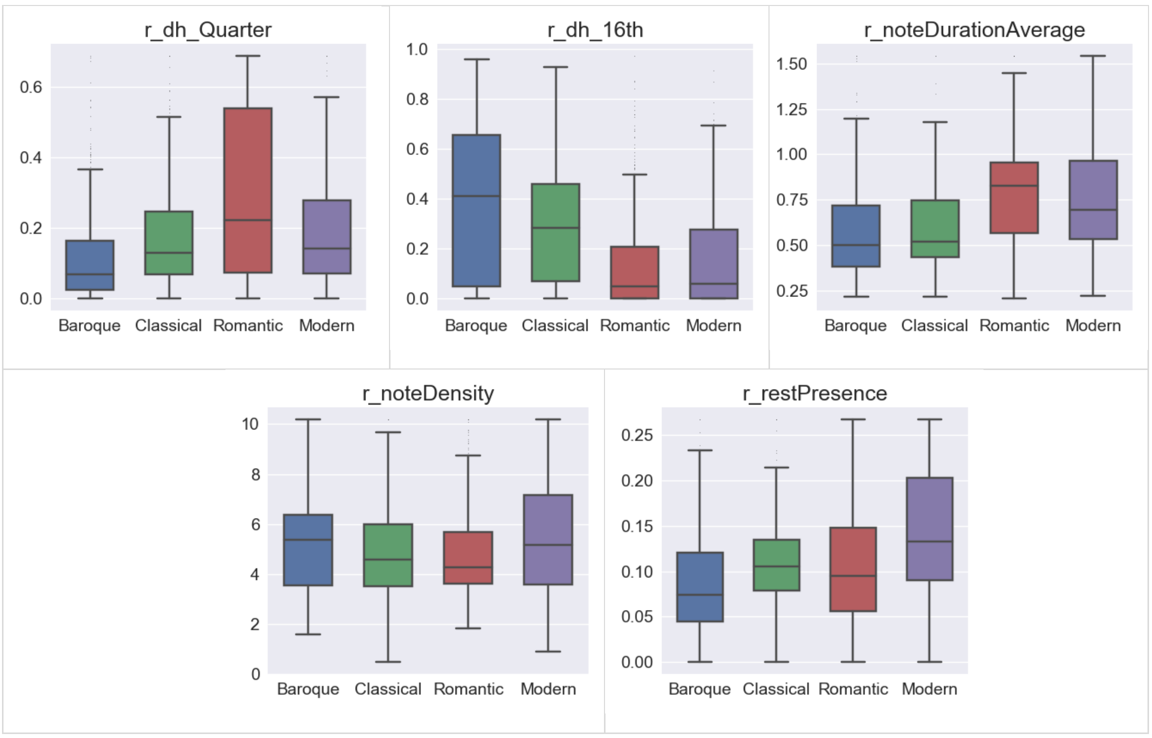


Figure 4.5: Distribution of most significant rhythm feature values

There does not appear to be a very significant difference in the note density among the musical periods, however the proportion of rest or silences in the music songs seems to be increasing in the periods, probably due to the rhythm being less strict in Romantic and Modern eras, allowing many changes in the tempo with the use of note rests.

Pitch

Figure 4.6 below, shows the distributions of the values for the maximum pitch, minimum pitch and range of pitches. It is observed that through the time periods there was an increasing tendency to use higher and also lower pitches of the keyboard. In the case of the Romantic period, there was more variety found in the range of maximum and minimum pitches, overlapping the distribution of the other periods.

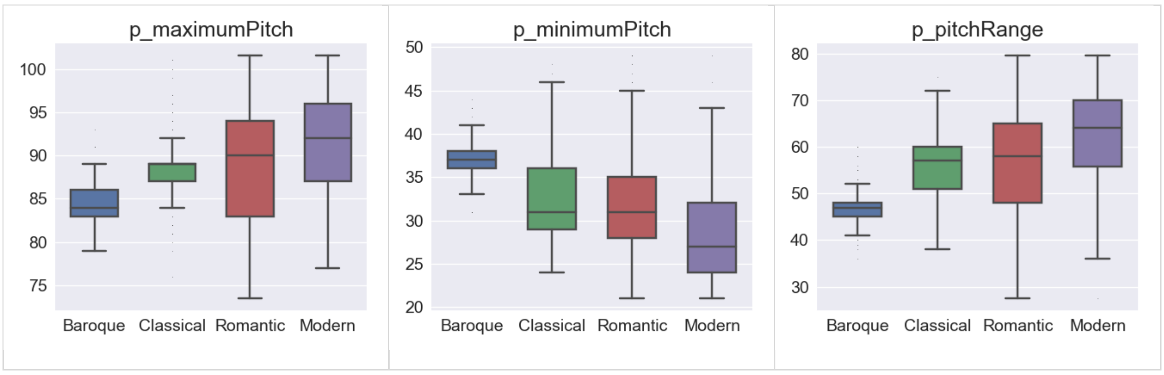


Figure 4.6: Distribution of most significant pitch feature values

A possible explanation is that the construction of the piano evolved in history and among the changes, one of them was the inclusion of more keys at both sides, around the mid-Baroque era (Giordano, 2015). Even though Bach, one of the most famous Baroque composers, had the opportunity to compose for the new instrument, it is said that he was never keen on it at his late age. Something similar could have happened with some other composers, even from the early Classical period, who never got used to the changes in the more modern pianos. It is also true that one of the main characteristics of the Romantic period is the expression and greater variety and originality in each piece, which could be achieved using a more dynamic range (Crocker, 1986).

Another pitch attribute that was found to be important, using the Random Forests, was the proportion of notes that are accidentals, or in other words, notes whose pitch class does not belong to the scale or mode indicated by the key signature. Checking the averages per class in the summary statistics, it looks like more accidentals have been used along the periods, with around 10% in Baroque and Classical, 15% in Romantic period, and 19% in Modern music. More accidentals means that the style was not so attached to the rules of tonality and attached to the key signature (Meyer, 1989).

Harmony

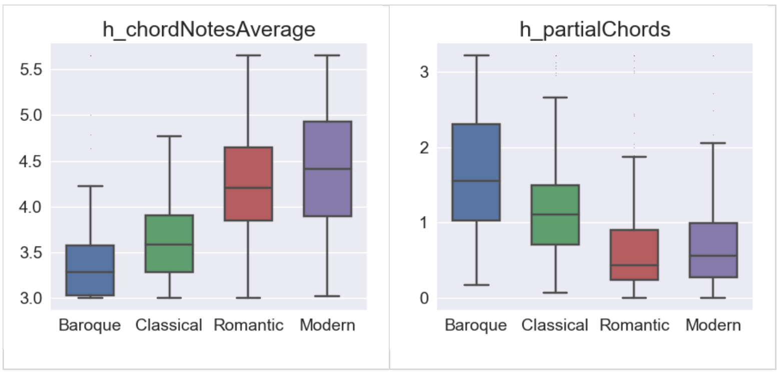


Figure 4.7: Distribution of harmony feature values in number of chord notes

Figure 4.7 above, depicts the distribution of the values for the information about the number of notes used in a chord in a music piece. The minimum number would be three and for the Baroque period, most of the pieces had a range from 3 to 3.5 of average in all the chords of a score. The evolution in the style in the following periods meant that more notes were added to the chords, suggesting that harmonies became more complicated than using major or minor triads, and moving to dissonant sounds and more complicated chords. The difference does not appear to be very relevant between the Romantic and Modern periods. In the feature that contained the average use of partial chords (two notes) per quarter/crotchet note, there is a contrast between the Baroque period and the rest.

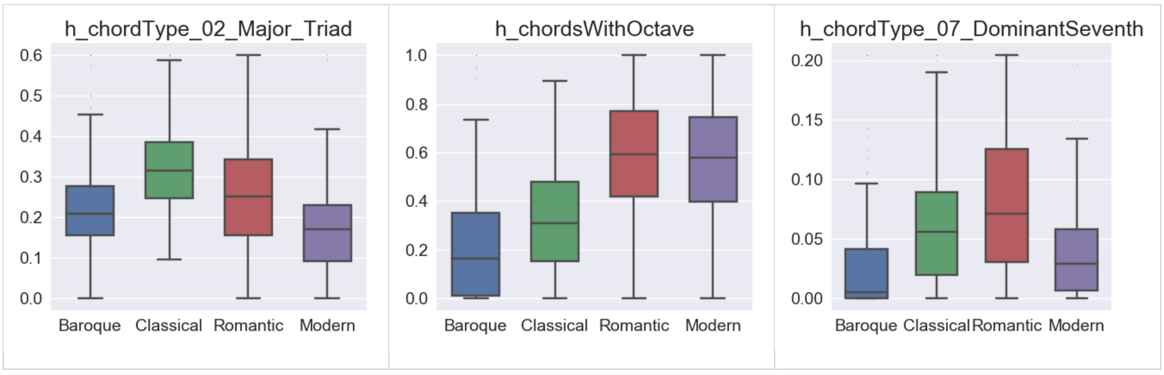


Figure 4.8: Distribution of harmony feature values in chord type formation

The other three harmony features in the ranking are shown in Figure 4.8. These features are related to the type of chords found in the score. Major and minor triads (not shown here), are the main chords used in the scores in all periods as it can be seen in the scales, with some scores having more than 40% of major triads. In this case there is not a linear trend in the changes, probably supported by the fact that harmonies in the Classical period were simpler, with composers trying to break from the previous style. After that the use of major triads decreased, with the Modern period showing the lowest values, maybe because more minor key modes were used. It is also interesting to see that the use of dominant seventh chords decreased in the Modern era.

Melody

Finally, the most significant melodic attributes with their distributions are presented in Figure 4.9. These attributes represent the fraction of major and minor second intervals, and the fraction of large intervals (higher than an octave).

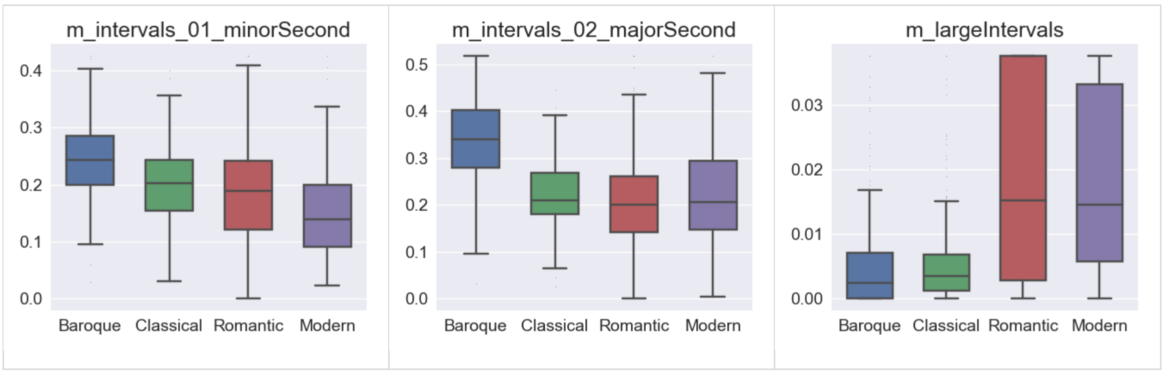


Figure 4.9: Distribution of most significant melody feature values

In this case there is a decreasing pattern in the use of second intervals, whether major or minor. A possible explanation is the shift away from tonality in Western classical music history, which became more pronounced in Romantic and especially Modern periods. The most common tonal scales are formed by a sequence of notes with one or two semitones of separation, so its use is more pronounced in works that respect the rules of tonality and the key signature. It is suggested that the use of these intervals could have decreased when using other types of intervals to form other scales.

Another feature found in the variable ranking of the melody category is the proportion of changes in the duration of a note along a melody, that will indicate the variability of tempo during a score.

4.4.2 SVM predictive model

The kernel method finally used for the SVM model was the Gaussian Radial Basis Function (RBF) which is often used in the literature when classifying music genres with symbolic music (Corrêa & Rodrigues, 2016; Hillewaere et al., 2009; Lidy et al., 2007).

A GridSearchCV method with python sklearn was used to determine the best settings for the SVM algorithm. With this method, using cross-validation, a model is created for all the combination of parameters, and the settings of the model with the best accuracy are returned. Due to the high computational demand of this process and the computing resources available, this method could not be performed completely and with all the set of parameters. At the end, the parameter values used for the predictive models were C = 100 and gamma set as auto.

4.5 Evaluation

A stratified 10-fold cross validation was used to evaluate the performance of each of the models trained. Using the stratified method, each of the folds will have the same proportion of each of the target classes as the original dataset. This is particularly important if some of the classes are imbalanced, as is the case with the 'Modern' class music.

Once the model is run, the predictions made by the model can be evaluated. By creating a cross-tabulation with the classes and the predictions, a confusion matrix can be plotted. This can be achieved with the Pandas method crosstab:

```
confusionMatrix = pd.crosstab(y, predictions)
```

The accuracy is calculated counting the number of correct precisions, divided by the number of records, or in other words the average of instances where the target class is equal to the predicted class.

```
scores = (y == predictions)
accuracy = scores.mean()
```

Different evaluation metrics, recall and precision, can be calculated using the confusion matrix, or simply using the metrics module of the python sklearn package.

```
Precision = metrics.precision_score(y, predictions)
Recall = metrics.recall_score(y, predictions)
```

The recall and precision are values for each of the classes. This way the models can be evaluated on a musical period basis, as opposed to other general metrics such as accuracy.

4.6 Conclusion

In this chapter, the creation of a dataset of music scores and the construction of features from them was detailed. The data preparation that was required with those extracted features was described and the first findings were shown. Feature importance for each category of features were obtained using Random Forest Tree algorithm. An investigation of the contribution of these attributes for each class was discussed.

The machine algorithm settings and the process to create the different predictive models was described, as well as how the different evaluation metrics were calculated.

The next chapter will present and evaluate the results of the models created with each set of feature categories, using confusion matrix and other evaluation metrics, such as recall and precision.

5. EVALUATION

5.1 Introduction

This chapter will evaluate the results of the models created in the experiment. Overall accuracy results will be presented and discussed. Further evaluation of the performance of the models in the classification of individual classes (musical periods) using confusion matrix and other metrics will then be outlined. The chapter will finish with a summary of the findings of this experiment.

5.2 Classification results

The predictions of all the test data in the cross-validation for each of the models was compared with the dataset of target classes and accuracies were calculated.

The overall results are presented in Table 5.1. The model using *harmony* features had the highest accuracy of the individual models with 74.4%, *pitch* features model reached 70%, and *rhythm* and *melody* had similar accuracy with 65.3% and 64.2% respectively. These results might not look impressive individually but it is worth noting that each of the categories represents a specific aspect of a musical composition. A music piece cannot be understood without any of these factors. For that reason, it was expected that the model using all the features would improve upon any of the individual models. Indeed, this proved to be true, with this model achieving 84.3% classification accuracy.

Model	**Accuracy**
Rhythm	0.65262
Pitch	0.70046
Harmony	0.74373
Melody	0.64237
All	0.84283

Table 5.1: Model Accuracy list

The accuracies of the models ranging from around 64 to 75 per cent, confirm that there were changes in each of these musical aspects through the classical music periods, to some extent. The initial analysis of the results would suggest that the changes in the harmony of a music piece were more distinctive in the music periods.

With the use of confusion matrices, an analysis of the results at a class level can be performed. The accuracy of the models only give an overall result, whereas confusion matrices and other metrics calculated from them, give information about how well each of the models performed to distinguish the different classes.

The confusion matrices for each of the models are shown in Tables 5.2, 5.3, 5.4 and 5.5. In the diagonal values, the True Positive (TP) outcomes can be observed, where the classifier was correct. The misclassifications are found in the rest of the values. For example, in Table 5.2, from all the Baroque pieces in the test set, the model predicted correctly as Baroque, 174 of them, and incorrectly classified 39 songs as Classical, 21 songs as Romantic and 3 songs as Modern.

Rhythm features model

		Predicted			
		Baroque	Classical	Romantic	Modern
Actual	Baroque	**174**	39	21	3
	Classical	40	**172**	27	10
	Romantic	30	31	**173**	26
	Modern	11	20	47	**54**

Table 5.2: Confusion matrix of the rhythm features model

Pitch features model

		Predicted			
		Baroque	Classical	Romantic	Modern
Actual	Baroque	**200**	24	13	0
	Classical	25	**186**	32	6
	Romantic	29	42	**167**	22
	Modern	5	22	43	**62**

Table 5.3: Confusion matrix of the pitch features model

Harmony features model

		Predicted			
		Baroque	Classical	Romantic	Modern
Actual	Baroque	**193**	28	14	2
	Classical	35	**191**	20	3
	Romantic	25	27	**187**	21
	Modern	11	5	34	**82**

Table 5.4: Confusion matrix of the harmony features model

Melody features model

		Predicted			
		Baroque	Classical	Romantic	Modern
Actual	Baroque	**179**	38	17	3
	Classical	38	**181**	27	3
	Romantic	27	48	**158**	27
	Modern	15	14	57	**46**

Table 5.5: Confusion matrix of the melody features model

When interpreting the confusion matrices, the number of records per class must be considered. If the four classes were completely balanced with 25% of records each, the outcomes could be comparable but in this study, the classes were slightly unbalanced, especially for the Modern period as explained in chapter 3. The four column values of the '*actual'* row in the confusion matrix for each period will add up to the number of records of that period.

With that in mind, some insights can be obtained by inspecting each of the tables. Looking at Table 5.2, it can be observed that the classifier using rhythm attributes had problems in classifying the Modern period music pieces as a similar number of songs were classified as Romantic (54 to 47), this suggests that there were not enough rhythm changes in the style of those two periods. Table 5.3 shows the outcomes of the predictive model using pitch features. It can be seen that the pitch features were a strong predictor category for the Baroque period, with an average good classification in Classical period. Table 5.4 shows the confusion matrix of the harmony features model. This model had the overall best accuracy and performed similarly well in the first three musical periods but with worse results in Modern class. The melody features model had the worst global accuracy largely because of the high number of misclassifications in the Romantic and Modern periods.

Looking globally at all the confusion matrices and especially at the misclassifications, a common pattern can be found in all the models. The error rate is gradually reduced the further apart the musical periods are in time. This is most obvious between the more extreme periods of Baroque and Modern classes. For example, in the pitch model, the results for the Baroque period were 200, 24, 13 and 0, for the four classes. The same decrease in misclassifications is found in all the models. This pattern confirms that the music style, although different, has a similarity in periods closer together in history (Crocker, 1986).

Confusion matrices give an overall view of the performance of the models but other measures can also be calculated from them. Using the total of records per class and the confusion matrix, recall and precision metrics were generated. For better comparison, the measures for each model have been compiled together.

Table 5.6 shows the recall measure for each model in each class. The term *accuracy class* is also used as synonymous of recall, as this represents the proportion of records in a class that are correctly classified.

Recall table

		Musical Period			
		Baroque	Classical	Romantic	Modern
Feature set	Rhythm	0.7342	0.6908	0.6654	0.4091
	Pitch	0.8439	0.7470	0.6423	0.4697
	Harmony	0.8143	0.7671	0.7192	0.6212
	Melody	0.7553	0.7269	0.6077	0.3485

Table 5.6: Recall table for each model and class (musical period)

Cells with green background represent the best accuracies for each of the classes, whereas red background cell represents the worse accuracy. Harmony features are the best predictors to identify Classical, Romantic and Modern music, and the classifier using pitch features had the best results in Baroque class and with an accuracy of 84.39% was also the overall best accuracy.

The table shows that the recall values for the Modern period are not very promising. This can indicate that the music style had small changes in each of the categories from previous periods, especially in melody, and a combination of all could improve the results.

All the models had their best results when classifying Baroque music, ranging from 73.4% to 84.3%. This is probably due to this musical period being the historical first in the dataset, therefore it would have only one period where styles could overlap. Another possible explanation is that Classicism in art, literature and as well in music, started as a dramatic rebellion against the style of the Baroque period with artists aiming for simplicity.

Another measure calculated from the confusion matrix, that is important in order to evaluate the models, is the precision. This will show the accuracy of a classifier when it makes a prediction for a certain class. It is possible that a classifier is very accurate with a class but just because it tends to give predictions of that class more times.

Similar to the recall table, Table 5.7 shows the precision of each of the models in the different classes with the best and worst values highlighted.

Precision table

		Musical Period			
		Baroque	Classical	Romantic	Modern
Feature set	Rhythm	0.6824	0.6565	0.6455	0.5806
	Pitch	0.7722	0.6788	0.6549	0.6889
	Harmony	0.7311	0.7610	0.7333	0.7593
	Melody	0.6911	0.6441	0.6100	0.5823

Table 5.7: Precision table per class (musical period)

The precision values in relation to the recall (or accuracy class) values are very similar, with the best scores coming from the same classifiers. When comparing the two tables there are a few considerations worth noting. In the Romantic and Modern periods, the precision was higher than their corresponding recall in all the models, as opposed to

Baroque and Classical where the values were lower, although still better than the other periods. It can be speculated that this is due to the slightly unbalanced classes and that the classifier will tend to predict the class with more records, therefore risking precision to improve accuracy. However, this is not the case in these models, as Romantic is the class with more records (29%), but achieves better precision than recall. It is interesting to see that Modern period does not get significant accuracy results but has considerably better precision. In the case of using pitch and harmony features, it has better precision than classifying Classical or Romantic music.

The following two tables show the predictive results of the model created using all the features. As mentioned earlier, this model had an accuracy of 84.28%, improving all the classifiers using sets of features.

Table 5.8 shows the confusion matrix and presents significantly better results than other classifiers.

Model with all features (rhythm, pitch , harmony and melody)

		Predicted			
		Baroque	Classical	Romantic	Modern
Actual	Baroque	**210**	15	12	0
	Classical	10	**218**	20	1
	Romantic	13	19	**213**	15
	Modern	4	2	27	**99**

Table 5.8: Confusion matrix of the model using all the features

Table 5.9 outlines the accuracy and precision values for each period. The ability of the algorithm to classify the songs, follows the pattern of the individual models previously evaluated in terms of accuracy per class.

	Accuracy	**Precision**
Baroque	0.8861	0.8861
Classical	0.8755	0.8583
Romantic	0.8192	0.7831
Modern	0.75	0.8609

Table 5.9: Accuracy and precision per class with all the features

This global model also performs better at classifying the Baroque and Classical periods, with 88.6% and 87.5%, respectively, suggesting that these periods had more factors of distinction than the other two. In the Baroque period, the global model improved the pitch features model by only 4.2%, confirming that the pitch is a strong predictor for this musical period. The classification of the Modern period with a 75% accuracy, improves significantly upon the results of the individuals models and it is the class that most benefits from the combination of categories. This indicates that the changes, although minor, came from very different aspects in music.

The accuracy of the models can also be seen at a composer level. It is worth noting that the classifier used the period as a target class to train the models and not the composer, and these results only show how correct the classifier was at predicting the period of these composers, not the composer himself. Therefore, these results must be treated with caution, and also because of the imbalance in the number of music pieces per composer, as can be seen in the Appendix section.

Period	**Composer**	**Rhythm**	**Pitch**	**Harmony**	**Melody**	**All**
Baroque	Domenico Scarlatti	60.0%	63.3%	65.0%	68.3%	80.0%
	Johann Sebastian Bach	80.9%	94.9%	91.1%	79.0%	93.6%
Classical	Franz Joseph Haydn	83.3%	75.0%	79.2%	66.7%	83.3%
	Ludwig van Beethoven	58.4%	74.3%	76.1%	69.0%	84.1%
	Muzio Clementi	64.7%	64.7%	64.7%	82.4%	76.5%
	Wolfgang Amadeus Mozart	83.1%	80.9%	83.1%	76.4%	95.5%
Romantic	Camille Saint-Saëns	30.0%	20.0%	20.0%	10.0%	40.0%
	Charles-Valentin Alkan	72.0%	72.0%	64.0%	44.0%	72.0%
	Edvard Grieg	47.8%	60.9%	69.6%	56.5%	73.9%
	Franz Liszt	33.3%	77.8%	66.7%	61.1%	83.3%
	Franz Schubert	77.8%	55.6%	94.4%	66.7%	100.0%
	Frederic Chopin	78.2%	63.2%	83.9%	79.3%	92.0%
	Johann Nepomuk Hummel	79.2%	79.2%	66.7%	37.5%	79.2%
	Johannes Brahms	90.0%	60.0%	70.0%	50.0%	70.0%
	Modest Mussorgsky	50.0%	81.3%	81.3%	56.3%	93.8%
Modern	Alexander Scriabin	30.8%	46.2%	61.5%	30.8%	53.8%
	Bela Bartok	9.1%	45.5%	54.5%	18.2%	72.7%
	Claude Debussy	57.1%	57.1%	61.9%	52.4%	85.7%
	Erik Satie	62.1%	37.9%	65.5%	51.7%	79.3%
	Maurice Ravel	30.0%	20.0%	70.0%	60.0%	90.0%
	Sergei Rachmaninoff	42.9%	38.1%	61.9%	14.3%	76.2%

Table 5.10: Accuracy of the models to predict the composer period.

Table 5.10 shows the accuracy of the different models in predicting the period by composer. The list only includes the composers from whom there was more than ten music pieces in the test set. The most characteristic composers from their periods, Bach, Mozart and Chopin, achieved significant results using all the features with 93.6%, 95.5% and 92% respectively. Despite Beethoven being one of the composers with greater representation in the data, the accuracy was worse, with 84.1%. This could be due to the fact that Beethoven was the predominant composer to connect the Classical and Romantic periods and his musical style would have changed in his later works. Looking at the accuracies of the individual models, it is possible to tentatively identify which musical properties of a composer's style are aligned with a particular musical period. For example, Liszt had a 83.3% accuracy with all sets, however, significantly less in the rhythm set (33.3%). This suggests that the rhythmic properties of his music were not typical of music in the Romantic style, and were more influenced by other styles. One could make these assumptions with more certainty if the data provided was much larger and with a similar representation of the composer's works.

5.3 Summary of findings

- According to the results of the experiment, the most important feature category to determine the musical style of Western classical music is harmony, achieving 74.4% accuracy. Although it is difficult to compare results with the literature, as other datasets were used, harmony alone using chord progressions was used in other papers with accuracies of 85.3% for 3 genres (Pérez-Sancho, Rizo, Kersten, & Ramirez, 2010).
- As expected, the combination of all the music properties improved the overall results achieving 84.3% accuracy. This is more noticeable in the Romantic and especially Modern styles, where this combination had a larger improvement compared to the other periods.
- In terms of specific attributes, when reviewing the importance of the features (provided by the RF variable ranking), the most important attributes have in common that they are related to tonality, for example, the use of pitch accidentals, chord types or melodic intervals. This confirms the evolution of the tonal system that has been studied by musicologists (Forte, 1988; Grout &

Palisca, 1996) and its importance in classifying musical styles. In addition, the range of note pitches used in a score, was a significant predictor of Baroque style. However, the evolution of the piano through history (Giordano, 2015), the instrument chosen for this dataset, could have influenced the importance of this attribute for the classification.

- A common pattern in all the models, including the one using all the music properties, is that the accuracy of the classifiers gradually decreased from the first to the last music period in history. The difficulty in classifying the music in the latter periods, suggests that there was more variety and changes inside those periods (Bennett, 1992).
- Analysis of the misclassifications in the models indicates that the error rate is higher in music styles that are closer historically. This confirms the gradual evolution in the music styles in history (Crocker, 1986).
- The ability of the classifier to predict the musical style at a composer level could be observed. The results showed that those composers whose musical period was identified with better accuracy were also very characteristic composers of their periods.

6. CONCLUSION

This chapter gives an overview of the research carried out for this dissertation. It will then describe the main contributions to the body of knowledge. Finally, the conclusion will outline the limitations of the study and give recommendations for future work.

6.1 Research overview

The purpose of this research was to determine the group of attributes, related to characteristics of music theory (rhythm, pitch, harmony and melody) that can better identify a music style in Western classical music pieces.

In order to carry out this experiment, a collection of music scores were downloaded from different sources, the main ones being Kern scores library and MuseScore online community. This was followed by the creation of a set of features extracted from these scores. The proposed features were based on the works examined in the literature review and the domain knowledge of the author.

Data preparation was carried out, including identifying and dealing with data quality issues, and applying data transformations to the data. For example, derived variables were created from the dataset, categorical variables were processed and normalisation was applied.

Finally, the features were divided in categories, and the SVM machine learning algorithm to classify the music pieces was implemented for each of them individually. The results were evaluated using a ten k-fold cross validation method. Additionally, a variable ranking was provided after obtaining the feature importance using a Random Forest algorithm.

6.2 Contributions

This research illustrates how computational analysis of music content could be used to find similarities among groups of music pieces that define a style, and consequently classify music material by its style.

The question that this research tried to answer was what musical property was more determinant in identifying the different musical styles. In particular, the research was focused on Western classical music and it was found that **harmony** was the most important category achieving an accuracy of 74.4% in classifying the four musical styles, Baroque, Classical, Romantic and Modern. Although, not the primary goal of the dissertation, it was also found that using all the feature categories, the accuracy improved to 84.3%.

The research confirms that the musical styles evolved gradually in the different periods, as stated in other studies (Crocker, 1986; Dalla Bella & Peretz, 2005; Grout & Palisca, 1996). Additionally, the changes in tonality through the years, reflected mainly in the harmony and melody properties, appears to be the most significant change that happened, as opposed to other changes such as variation of rhythm.

Another contribution of this research is the collection of a dataset of piano music scores and the implementation to extract features from them, based on the review of music theory and proposed attributes in other research papers, such as the work of McKay and Fujinaga (Mckay & Fujinaga, 2006). This dataset could be extended by adding more music scores and more features to perform other type of analysis, such as classification by composer, forms, or evolution of a particular style.

6.3 Limitations of the research

An important limitation of this research is the dataset used for the experiment. It was not possible to collect the most representative dataset of all the music styles. In the four classes used for the classification, the Modern class presented 15% of the scores, as opposed to a perfectly balanced 25%. This imbalance, could have influenced the results as less information was available to determine the characteristics of this period, compared to the other styles. Also, the number of composers in each music period was not homogeneous. There were less different composers and with more music pieces in the Baroque and Classical period, and a greater variety of composers but with less records, in the Romantic and especially Modern music. Although all the composers in a period would share similarities in the style, their own particular style, despite being difficult to assess, would influence the capability of the classifier.

The classes or musical periods were assigned according to the composer. Relying in the composer as a basis to assign the label of the musical style or period is a common practice and has been used frequently in the literature (Rodriguez Zivic et al., 2013; Schaab & Weiss, 2015). However, this method can carry some issues as the styles are not clearly defined. The styles or periods frequently overlap, individual music pieces could belong to other styles, even if written by the same composer, and depending on locations, the styles might have taken more time to be established. (McKay & Fujinaga, 2006)

The other limitation in the research is the machine learning algorithm tuning. A significant period of time of this research was dedicated to construction of the feature set from the raw music score files that would be used to determine the music styles. Therefore, less time was available to find the optimal parameters to tune the machine learning SVM algorithm. This was aggravated by the fact that testing multiple combinations of parameters was computationally intensive. Additionally, more time could have been spent in testing other machine learning algorithms, such as neural networks or even deep learning. However, the main objective of the research was to compare different feature sets (musical properties) and not machine learning algorithms.

6.4 Future work and recommendations

A larger set of music scores files could be constructed, including music from other instruments and orchestral music. If not possible, an alternative could be to resample the current catalogue of music scores, by splitting each score by a number of seconds or measures. With more data files, the model could generalise better by finding more similarities in the musical styles.

Regarding the music styles defined by the music periods, other taxonomies could be used instead. According to Roy Bennett, the Modern period (20th century) had more changes in style than any other period in history, and subgenres such as impressionism, expressionism or serialism existed at the same time (Bennett, 1992). By dividing the periods in more precise groups, more insights could be gathered and results could improve too.

More future work should be dedicated to the improvement of the general classifier, using all the feature categories, by performing the following actions:

- Use the information about feature importance as a feature selection step that can make the model more accurate by removing irrelevant features.
- Review the feature extraction of music scores process, focusing on the categories that had poor results, melody and rhythm. For example, adding a

melodic similarity approach (Müllensiefen & Frieler, 2006) or including rhythm pattern extractions similar to the ones proposed in (Lev et al., 2011).

- In that aspect, more research of the music theory and analysis has to be done, in order to construct more specific features that account for chord progressions, arpeggiated harmonies, or rhythmic syncopations.
- If the computing resources are available, optimize the algorithm parameters of the SVM algorithm by testing multiple combinations of settings, and also experiment with other machine learning algorithms, such as neural networks, or even deep learning techniques
- Try using an ensemble method algorithm, using the results of each of the individual models with sets of features (rhythm, pitch, harmony and melody). The final prediction will be calculated by using a majority-based result of the individual models.

It would also be interesting to see how the same set of features and classifier algorithm would perform in other type of classifications such as composer, or using other types of music rather than Classical music.

BIBLIOGRAPHY

Bainbridge, D., & Bell, T. (2001). The Challenge of Optical Music Recognition. *Computers and the Humanities*, *35*, 95–121.

Ben-Hur, A., & Weston, J. (2010). A user's guide to support vector machines. *Methods in Molecular Biology (Clifton, N.J.)*, *609*, 223–239. https://doi.org/10.1007/978-1-60327-241-4_13

Bennett, R. (1992). *Investigating Musical Styles*. Cambridge University Press.

Borovicka, T., Jr., M. J., Kordik, P., & Jirina, M. (2012). Selecting Representative Data Sets. In A. Karahoca (Ed.), *Advances in Data Mining Knowledge Discovery and Applications*. Rijeka: InTech. https://doi.org/10.5772/50787

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Cawley, G. C., & Talbot, N. L. C. (2010). On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, *11*, 2079–2107.

Corrêa, D. C., & Rodrigues, F. A. (2016). A survey on symbolic data-based music genre classification. *Expert Systems With Applications*, *60*, 190–210. https://doi.org/10.1016/j.eswa.2016.04.008

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1023/A:1022627411411

Crocker, R. L. (1986). *A history of musical style*. Dover Publications.

Cuthbert, M. S., & Ariza, C. (2010). music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *11th International Society for Music Information Retrieval Conference (ISMIR 2010)2* (pp. 637–642).

Dalla Bella, S., & Peretz, I. (2005). Differentiation of classical music requires little learning but rhythm. *Cognition*, *96*(2), B65–B78.

Ferkova, E., Ždimal, M., & Šidlik, P. (2007). Chordal Evaluation in MIDI-Based Harmonic Analysis: Mozart, Schubert, and Brahms. *Computing in Musicology*, 172–186.

Flom, R., Gentile, D. A., & Pick, A. D. (2008). Infants' discrimination of happy and sad music. *Infant Behavior and Development*, *31*(4), 716–728. https://doi.org/10.1016/j.infbeh.2008.04.004

Forte, A. (1988). Pitch-Class Set Genera and the Origin of Modern Harmonic Species. *Journal of Music Theory*, *32*(2), 187–270. https://doi.org/10.2307/843436

Fradera, J. J. (2003). *El lenguaje musical*. Robinbook.

Fu, Z., Lu, G., Ting, K. M., & Zhang, D. (2011). A Survey of Audio-Based Music Classification and Annotation. *IEEE TRANSACTIONS ON MULTIMEDIA*, *13*(2). https://doi.org/10.1109/TMM.2010.2098858

Genuer, R., Poggi, J.-M., & Tuleau-Malot, C. (2011). Variable selection using Random Forests. In *Procedia Environmental Sciences* (Vol. 3, pp. 44–49). https://doi.org/10.1016/j.proenv.2011.02.009

Giordano, N. (2015). Evolution of the piano. *The Journal of the Acoustical Society of America*, *138*(3), 1912–1912. https://doi.org/10.1121/1.4934012

Good, M. (2001). MusicXML: An internet-friendly format for sheet music. *XML Conference and Expo*, 1–12. https://doi.org/10.1.1.118.5431

Grachten, M., Arcos, J. L., & Mántaras, R. L. De. (2004). Melodic Similarity: Looking for a Good Abstraction Level. *Proceedings of the 5th International Society for Music Information Retrieval*.

Grout, D. J., & Palisca, C. V. (1996). A history of Western music. *A History of Western Music.*, (Ed. 5).

Guyon, I., Elisseeff, A., & De, A. M. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, *3*, 1157–1182.

Hadjeres, G., Pachet, F., & Nielsen, F. (2016). DeepBach: a Steerable Model for Bach Chorales Generation. Retrieved from http://arxiv.org/abs/1612.01010

Han, J., & Kamber, M. (2011). *Data Mining: Concepts and Techniques*. *Elsevier* (Vol. 12). https://doi.org/10.1007/978-3-642-19721-5

Hillewaere, R., Manderick, B., & Conklin, D. (2009). Global Feature Versus Event Models for Folk Song Classification. *10th International Society for Music Information Retrieval Conference (ISMIR 2009)*, (Ismir), 729–733.

Huron, D. (1997). Beyond MIDI. In E. Selfridge-Field (Ed.) (pp. 375–401). Cambridge, MA, USA: MIT Press.

Isikhan, C., & Ozcan, G. (2008). A survey of melody extraction techniques for music information retrieval. In *Proceedings of 4th Conference on Interdisciplinary Musicology (SIM'08)*.

Jackendoff, R., & Lerdahl, F. (2006). The capacity for music: What is it, and what's special about it? *Cognition*, *100*(1), 33–72.

Kaminskas, M., & Ricci, F. (2012). Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, *6*(2–3), 89–119. https://doi.org/10.1016/j.cosrev.2012.04.002

Karydis, I. (2006). Symbolic music genre classification based on note pitch and duration. In *Lecture Notes in Computer Science* (Vol. 4152 LNCS, pp. 329–338).

Kelleher, J. D., Namee, B. Mac, & D'Arcy, A. (2015). *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT Press*.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Proc. of IJCAI'95*, *14*, 1137–1145.

Lebar, J., Chang, G., & Yu, D. (2010). Classifying Musical Scores by Composer: a machine learning approach. *Proceedings of 3rd International Workshop on Machine Learning and Music - MML '10*, 37–40.

Lee, J. H., & Downie, J. S. (2004). Survey of Music Information Needs, Uses, and Seeking Behaviours: Preliminary Findings. In *ISMIR 2004* (pp. 441–446). https://doi.org/10.1109/ISM.2009.123

León, P. J. P. de, & Iñesta, J. M. (2004). Musical Style Classification from Symbolic Data: A Two-Styles Case Study (pp. 167–178). https://doi.org/10.1007/978-3-540-39900-1_15

Lev, F., Groult, R., Arnaud, G., Cyril, S., Picardie, D., Verne, J., & Giraud, M. (2011). Rhythm extraction from polyphonic symbolic music. *Information Retrieval*, (Ismir), 375–380.

Lidy, T., Rauber, A., Pertusa, A., & Iñesta, J. M. (2007). Improving genre classification by combination of audio and symbolic descriptors using a transcription system. *Proceedings of the 8th International Symposium on Music Information Retrieval, Vienna, Austria*, 1–6.

Lu, L., Liu, D., & Zhang, H. J. (2006). Automatic mood detection and tracking of music audio signals. In *IEEE Transactions on Audio, Speech and Language Processing* (Vol. 14, pp. 5–18). https://doi.org/10.1109/TSA.2005.860344

Marsden, A. (2010). Schenkerian analysis by computer: A proof of concept. *Journal of New Music Research*, *39*(3), 269–289.

Mckay, C., & Fujinaga, I. (2004). Automatic Genre Classification Using High-Level Musical Feature Sets. *Proceedings of the 5th International Conference on Music Information Retrieval*, 525–530.

Mckay, C., & Fujinaga, I. (2005). The bodhidharma system and the results of the mirex 2005 symbolic genre classification contest. *International Symposium/Conference on Music Information Retrieval*, 1–4.

Mckay, C., & Fujinaga, I. (2006). jSymbolic: A Feature Extractor for MIDI Files. *Proceedings of the International Computer Music Conference.*, 302–305.

McKay, C., & Fujinaga, I. (2006). Musical genre classification: Is it worth pursuing and how can it be improved? *ISMIR*, 101–106.

Meyer, L. B. (1989). *Style and Music: History, Theory and Ideology*. University of Chicago Press.

Müllensiefen, D., & Frieler, K. (2006). Evaluating Different Approaches to Measuring the Similarity of Melodies. *Data Science and Classification*, 299–306. https://doi.org/10.1007/3-540-34416-0

Pardo, B., Shifrin, J., & Birmingham, W. (2004). Name that tune: A pilot study in finding a melody from a sung query. *Journal of the American Society for Information Science and Technology*, *55*(4), 283–300.

Pérez-Sancho, C., Rizo, D., Kersten, S., & Ramirez, R. (2010). Genre classification of music by tonal harmony. *Intelligent Data Analysis*, *14*(5), 533–545. https://doi.org/10.3233/IDA-2010-0437

Ponce de Léon, P., & Carlos, P. (2004). A shallow description framework for musical style recognition. *Structural, Syntactic, and Statistical Pattern Recognition,* (August), 876–884. https://doi.org/10.1007/978-3-540-27868-9

Ponce de León, P. J., Iñesta, J. M., & Rizo, D. (2008). Mining Digital Music Score Collections : Melody Extraction and Genre Recognition. In *Pattern Recognition Techniques, Technology and Applications, Peng-Yeng Yin (Ed.).* InTech. https://doi.org/10.5772/6259

Ringwalt, D., Dannenberg, R. B., & Russell, A. (2015). Optical Music Recognition for Interactive Score Display. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 95–98.

Rodriguez Zivic, P. H., Shifres, F., & Cecchi, G. A. (2013). Perceptual basis of evolving Western musical styles. *Proceedings of the National Academy of Sciences*, *110*(24), 10034–10038. https://doi.org/10.1073/pnas.1222336110

Roig, C., Tardón, L. J., Barbancho, I., & Barbancho, A. M. (2014). Automatic melody composition based on a probabilistic model of music style and harmonic rules. *Knowledge-Based Systems*, *71*, 419–434.

Sapp, C. S. (2005). Online Database of Scores in the Humdrum File Format. *ISMIR 2005*, 664–665.

Schaab, M., & Weiss, C. (2015). On the impact of key detection performance for identifying classical music styles. *16th International Society for Music Information Retrieval Conference (ISMIR 2015)*, 45–51.

Sturm, B. L. (2012). A Survey of Evaluation in Music Genre Recognition. *International Workshop on Adaptive Multimedia Retrieval*, 29–66. https://doi.org/10.1007/978-3-319-12093-5

Taylor, E. R. (1991). *The AB guide to music theory*. Associated Board of the Royal Schools of Music.

Thom, B., Spevak, C., & Höthker, K. (2002). Melodic segmentation: Evaluating the performance of algorithms and musical experts. *Proceedings of the International Computer Music Conference, 2002*(12), 65–72.

Typke, R., Wiering, F., & Veltkamp, R. C. (2005). A Survey of Music Information Retrieval Systems. *Proceedings of 6th International Conference on Music Information Retrieval (ISMIR 2005)*, (January).

Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, *10*(5), 292–302. https://doi.org/10.1109/TSA.2002.800560

Tzanetakis, G., Ermolinskyi, A., & Cook, P. (2003). Pitch Histograms in Audio and Symbolic Music Information Retrieval. *Journal of New Music Research*, *32*(2), 143–152. https://doi.org/10.1076/jnmr.32.2.143.16743

Van Kranenburg, P. (2006). Composer Attribution by Quantifying Compositional Strategies. *Proceedings of International Symposium on Music Information Retrieval*, 375–376.

Xu, C., Maddage, N., & Shao, X. (2003). Musical genre classification using support vector machines. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03)* (Vol. 5, p. V-429-32 vol.5).

Appendix A: Number of scores by composer

Period	Composer	Scores
Baroque	Dieterich Buxtehude	2
	Domenico Scarlatti	60
	Francois Couperin	2
	Georg Phillipp Teleman	1
	George Frideric Handel	9
	Henry Purcell	4
	Jean-Philippe Rameau	2
	Johann Sebastian Bach	157
Classical	Carl Maria von Weber	1
	Carl Philipp Emanuel Bach	4
	Franz Joseph Haydn	24
	Ignaz Pleyel	1
	Ludwig van Beethoven	113
	Muzio Clementi	17
	Wolfgang Amadeus Mozart	89
Romantic	Camille Saint-Saëns	10
	Charles-Valentin Alkan	25
	Edvard Grieg	23
	Edward MacDowell	9
	Felix Mendelssohn	5
	Franz Liszt	18
	Franz Schubert	18
	Frederic Chopin	87
	Georges Bizet	2
	Jacques Offenbach	1
	Johann Nepomuk Hummel	24
	Johann Strauss II	2
	Johannes Brahms	10
	John Field	1
	Mikhail Glinka	1
	Modest Mussorgsky	16
	Pyotr Ilyich Tchaikovsky	3
	Robert Schumann	5
Modern	Alexander Scriabin	13
	Anton Webern	2
	Aram Khachaturian	1
	Bela Bartok	11
	Claude Debussy	21
	Dmitri Shostakovich	4
	Erik Satie	29
	Francis Poulenc	1
	Igor Stravinsky	1
	Jacques Ibert	1
	Jean Sibelius	2
	Josef Suk	8
	Maurice Ravel	10
	Sergei Prokofiev	7
	Sergei Rachmaninoff	21

Appendix B: Summary statistics

Rhythm attributes

Field	type	count	unique	min	max	mean	std	Q1	Median	Q3
r_dh_16th	float64	878	710	0	0.969432	0.254469	0.26157	0.011272	0.172016	0.460202
r_dh_32nd	float64	878	331	0	0.741085	0.033804	0.09558	0	0	0.011205
r_dh_Breve	float64	878	84	0	0.020408	0.000301	0.001537	0	0	0
r_dh_Dotted_16th	float64	878	109	0	0.216535	0.001862	0.010456	0	0	0
r_dh_Dotted_32nd	float64	878	11	0	0.013514	4.22E-05	0.000554	0	0	0
r_dh_Dotted_Eighth	float64	878	509	0	0.330163	0.014804	0.035089	0	0.002279	0.014167
r_dh_Dotted_Half	float64	878	528	0	0.638554	0.01467	0.037386	0	0.00266	0.014572
r_dh_Dotted_Quarter	float64	878	682	0	0.462279	0.021605	0.037176	0.001442	0.008699	0.024255
r_dh_Dotted_Whole	float64	878	182	0	0.114458	0.001276	0.00631	0	0	0
r_dh_Eighth	float64	878	844	0	0.963156	0.3163	0.214862	0.154112	0.282166	0.4465
r_dh_Half	float64	878	731	0	0.613757	0.039731	0.063239	0.004191	0.017892	0.048052
r_dh_Quarter	float64	878	860	0	0.939673	0.204446	0.201312	0.05315	0.125272	0.306047
r_dh_Whole	float64	878	403	0	0.166189	0.006145	0.016206	0	0	0.004226
r_distanceTwoMostFrequent	float64	878	18	1.166667	24	2.397988	1.358143	2	2	2
r_dupleMeter	bool	878	2							
r_durationRange	float64	878	102	0.002083	0.5	0.063368	0.060436	0.025	0.041667	0.083333
r_expressionsPresence	float64	878	393	0	0.13253	0.003479	0.009174	0	0	0.002936
r_graceNotePresence	float64	878	400	0	0.214286	0.007735	0.018269	0	0	0.008206
r_maximumDuration	float64	878	68	0.5	42	5.499687	4.626761	3	4	6.5
r_minimumDuration	float64	878	28	0.016667	1	0.227281	0.145995	0.125	0.25	0.25
r_mostCommonNoteDuration	float64	878	14	0.083333	3	0.50764	0.334065	0.25	0.5	0.5
r_mostCommonNotePresence	float64	878	874	0.152406	0.995392	0.553728	0.15312	0.438329	0.540705	0.663331
r_nonBasicDurationsPresence	float64	878	771	0	0.997409	0.090543	0.17761	0.002914	0.019022	0.067941
r_noteDensity	float64	878	862	0.484375	16.96629	5.053375	2.093414	3.55869	4.703227	6.203327
r_noteDurationAverage	float64	878	877	0.204098	3.385542	0.684362	0.324875	0.441686	0.617853	0.881087
r_noteDurationStdDev	float64	878	878	0.043083	1.761009	0.552827	0.294498	0.34464	0.479046	0.695824
r_numberOfDistinctDurations	int64	878	34	2	43	10.38952	5.215622	7	9	13
r_restPresence	float64	878	858	0	0.612676	0.110172	0.075264	0.060114	0.096631	0.142722
r_timeSignatureChanges	int64	878	18	0	48	0.731207	2.886797	0	0	0
r_tripleMeter	bool	878	2							
r_twoMostCommonNotePresence	float64	878	866	0.291444	1	0.793841	0.123428	0.718348	0.808562	0.89059

Pitch attributes

Field	type	count	unique	min	max	mean	std	Q1	Median	Q3
p_accidentals	float64	878	870	0	0.798742	0.134178	0.088405	0.076168	0.12059	0.167468
p_distanceStrongestPitchClasses	int64	878	6	1	6	4.473804	1.013533	4	5	5
p_distanceStrongestPitches	int64	878	26	1	36	7.653759	5.772654	4	5	12
p_keySignatureChanges	int64	878	11	0	24	0.6082	1.619163	0	0	0
p_maximumPitch	int64	878	40	66	108	87.98975	6.218275	84	88	91
p_minimumPitch	int64	878	33	21	55	33.12984	5.683118	29	33	37
p_mostCommonPitchPresence	float64	878	852	0.035593	0.319835	0.085438	0.028934	0.067597	0.079091	0.095399
p_numberOfPitchClasses	int64	878	6	7	12	11.5615	0.983431	12	12	12
p_numberOfPitches	int64	878	66	15	85	46.84966	12.2468	39	45	55
p_ph01	float64	878	863	0.090032	0.472185	0.193864	0.043311	0.164401	0.18707	0.214286
p_ph02	float64	878	853	0.089639	0.267624	0.161096	0.025837	0.145187	0.156762	0.175482
p_ph03	float64	878	840	0.085561	0.209302	0.135392	0.018542	0.123733	0.135756	0.146342
p_ph04	float64	878	850	0.062663	0.16955	0.116861	0.016468	0.104698	0.117629	0.129417
p_ph05	float64	878	842	0.049505	0.14359	0.102502	0.016704	0.090931	0.102154	0.114838
p_ph06	float64	878	837	0.030788	0.134788	0.088346	0.016012	0.078178	0.086979	0.098277
p_ph07	float64	878	855	0.014493	0.121429	0.072474	0.015742	0.063313	0.072345	0.082083
p_ph08	float64	878	847	0	0.087452	0.044656	0.016267	0.032855	0.045037	0.056171
p_ph09	float64	878	826	0	0.080386	0.031681	0.01707	0.018681	0.030897	0.042532
p_ph10	float64	878	795	0	0.080386	0.024004	0.015928	0.011982	0.022463	0.033844
p_ph11	float64	878	758	0	0.080386	0.017452	0.014074	0.006337	0.015516	0.024915
p_ph12	float64	878	664	0	0.074792	0.01167	0.012243	0.001786	0.008633	0.017094
p_pitchAverage	float64	878	878	48.66512	76.90987	63.53541	3.279711	61.88495	63.59869	65.50171
p_pitchRange	int64	878	56	26	87	54.85991	10.17194	47	54	60
p_pitchStd	float64	878	878	5.539779	23.43379	11.09123	2.019119	9.736474	10.75334	12.04754
p_strengthOfTopPitches	float64	878	680	0.186047	1	0.859224	0.118708	0.791784	0.888889	0.95166

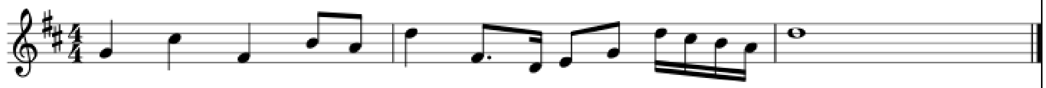
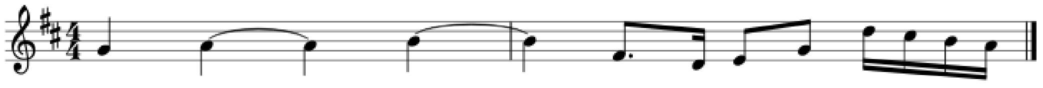
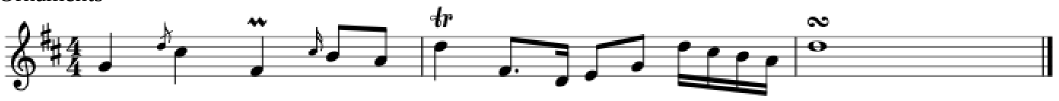
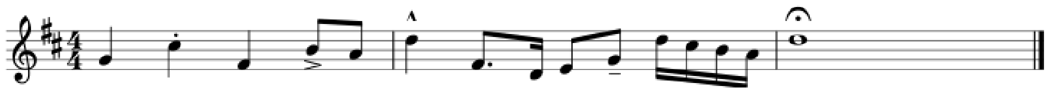
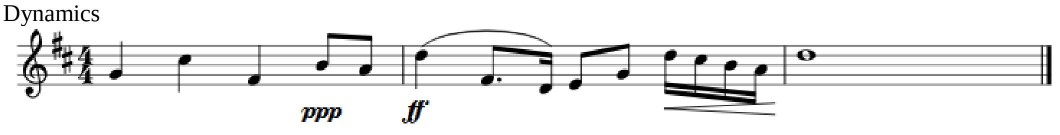
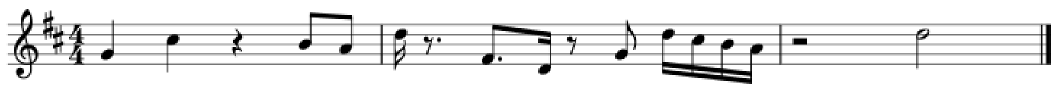
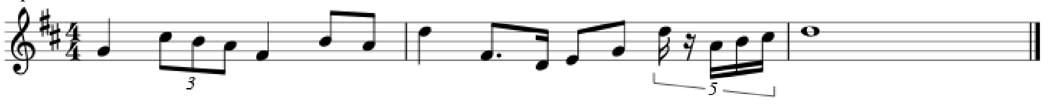
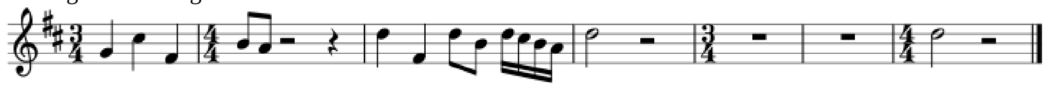
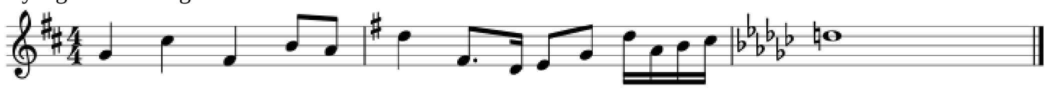
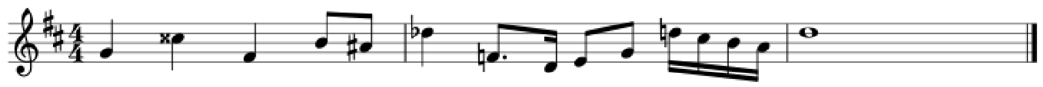
Harmony attributes

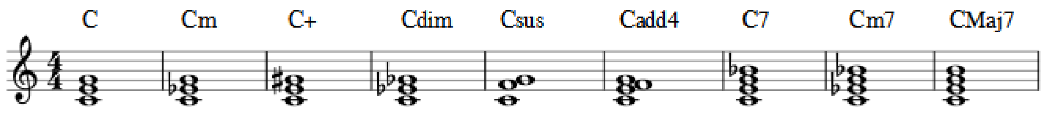
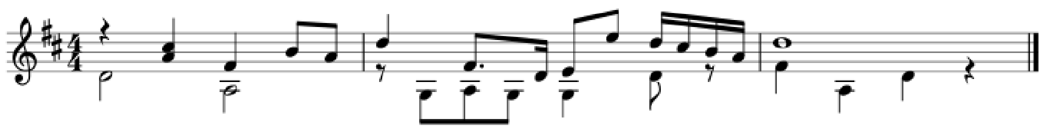
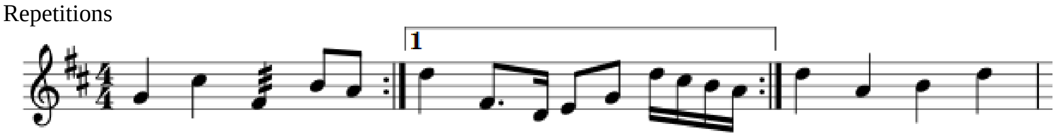
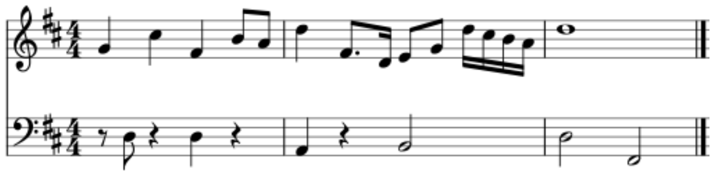
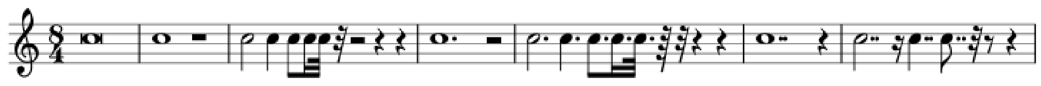
field	type	count	unique	top	freq	min	max	mean	std	Q1	Median	Q3
h_chordDensity	float64	841	808			0.006579	5.590503	1.095873	0.728282	0.581976	0.940789	1.434
h_chordDurationAverage	float64	841	788			0.10451	6	0.513703	0.356624	0.304965	0.451426	0.632867
h_chordNotesAverage	float64	841	766			3	8.545455	3.90471	0.773605	3.330661	3.773006	4.257642
h_chordNotesStd	float64	837	788			0	3.295601	0.704208	0.408127	0.476549	0.672886	0.919167
h_chordPitchClassesAverage	float64	841	702			3	4.174583	3.265075	0.223244	3.090226	3.234694	3.386691
h_chordPitchClassesStd	float64	837	756			0	1.079185	0.412306	0.213524	0.298091	0.445498	0.55095
h_chordType_01_Minor_Triad	float64	841	712			0	1	0.148224	0.099143	0.083264	0.135838	0.196998
h_chordType_02_Major_Triad	float64	841	747			0	1	0.263434	0.151246	0.165605	0.242588	0.33871
h_chordType_03_Diminished_Triad	float64	841	637			0	0.389706	0.069357	0.063331	0.020548	0.057692	0.101449
h_chordType_04_Augmented_Triad	float64	841	412			0	0.5	0.010794	0.023732	0	0.003861	0.014286
h_chordType_05_OtherTriads	float64	841	747			0	1	0.273614	0.159041	0.158824	0.253968	0.368132
h_chordType_06_MinorSeventh	float64	841	469			0	0.317073	0.020827	0.032513	0	0.008772	0.02765
h_chordType_07_DominantSeventh	float64	841	593			0	0.555556	0.058796	0.067709	0.006211	0.041257	0.085366
h_chordType_08_MajorSeventh	float64	841	403			0	0.324561	0.012165	0.021374	0	0.003378	0.017101
h_chordType_09_Other4Chords	float64	841	667			0	0.617021	0.114832	0.099014	0.030769	0.102041	0.170418
h_chordType_10_ComplexChords	float64	841	415			0	0.350449	0.027956	0.051624	0	0.00406	0.034014
h_chordsWithOctave	float64	841	721			0	1	0.412605	0.284794	0.176471	0.401361	0.615385
h_harmonic_Interval_01_m2	float64	878	804			0	0.104938	0.012866	0.011353	0.00533	0.01092	0.017269
h_harmonic_Interval_02_M2	float64	878	859			0	0.181931	0.048111	0.022598	0.03294	0.045588	0.059856
h_harmonic_Interval_03_m3	float64	878	870			0	0.311721	0.137425	0.03803	0.1123	0.136937	0.160719
h_harmonic_Interval_04_M3	float64	878	863			0	0.290698	0.125301	0.030364	0.107826	0.123081	0.139951
h_harmonic_Interval_05_P4	float64	878	861			0	0.217822	0.098534	0.024871	0.08379	0.09775	0.111111
h_harmonic_Interval_06_TT	float64	878	860			0	0.172783	0.055101	0.024398	0.038828	0.054235	0.067861
h_harmonic_Interval_07_P5	float64	878	861			0	0.261473	0.114828	0.029846	0.096153	0.112816	0.132087
h_harmonic_Interval_08_m6	float64	878	861			0	0.165354	0.077984	0.021445	0.064372	0.078063	0.089883
h_harmonic_Interval_09_M6	float64	878	864			0	0.38374	0.101826	0.028582	0.085663	0.099962	0.117285
h_harmonic_Interval_10_m7	float64	878	862			0	0.12911	0.049601	0.018972	0.037057	0.048496	0.060821
h_harmonic_Interval_11_M7	float64	878	823			0	0.310219	0.018591	0.019108	0.00982	0.016058	0.023164
h_harmonic_Interval_12_P8	float64	878	867			0	1	0.159831	0.070192	0.114642	0.148561	0.19041
h_mostCommonChordSet	float64	841	710			0.023083	1	0.162565	0.11606	0.09369	0.134503	0.193798
h_partialChords	float64	878	832			0	5.443299	1.110791	0.87768	0.4364	0.936846	1.546606
h_topVerticalInterval	object	878	10	P8	379							
h_topVerticalIntervalPresence	float64	878	867			0.112957	1	0.189412	0.056419	0.15649	0.176538	0.205378
h_uniqueChordDurations	float64	841	28			1	53	5.302021	4.438641	3	4	7
h_uniqueChords	float64	841	751			0.09375	1	0.603707	0.194037	0.458333	0.589744	0.74359

Melody attributes

field	type	count	unique	top	freq	min	max	mean	std	Q1	Median	Q3
m_directionDown	float64	878	856			0.171875	0.693642	0.451846	0.073813	0.41827	0.4598	0.498382
m_directionSame	float64	878	844			0	0.604082	0.118571	0.105819	0.044373	0.09086	0.161706
m_directionUp	float64	878	853			0.146939	0.720131	0.429583	0.070557	0.393047	0.434508	0.470271
m_intervals_01_minorSecond	float64	878	863			0	0.469269	0.201354	0.079944	0.146253	0.201895	0.257749
m_intervals_02_majorSecond	float64	878	868			0	0.833333	0.249323	0.106755	0.178783	0.239607	0.314055
m_intervals_03_minorThird	float64	878	860			0	0.344464	0.106702	0.054108	0.066667	0.100424	0.137308
m_intervals_04_majorThird	float64	878	846			0	0.357664	0.071932	0.038634	0.044254	0.065403	0.092611
m_intervals_05_perfectFourth	float64	878	842			0	0.319444	0.078533	0.038558	0.053041	0.074257	0.097841
m_intervals_06_Tritone	float64	878	759			0	0.142857	0.018502	0.01957	0.005607	0.013701	0.024104
m_intervals_07_perfectFifth	float64	878	834			0	0.324855	0.050344	0.036384	0.025228	0.041301	0.064728
m_intervals_08_minorSixth	float64	878	769			0	0.264045	0.021748	0.023783	0.007588	0.014985	0.028249
m_intervals_09_majorSixth	float64	878	775			0	0.302198	0.02301	0.025739	0.007583	0.015848	0.030397
m_intervals_10_minorSeventh	float64	878	735			0	0.280757	0.015365	0.01869	0.004373	0.010586	0.020294
m_intervals_11_majorSeventh	float64	878	608			0	0.253968	0.00609	0.014388	0	0.003127	0.007709
m_intervals_12_Octave	float64	878	822			0	0.611285	0.038526	0.046168	0.012247	0.024769	0.046566
m_largeIntervals	float64	878	674			0	0.266667	0.015961	0.027791	0.001399	0.00517	0.015879
m_melodicContourChange	float64	878	865			0.199262	0.997685	0.577991	0.101378	0.511519	0.577406	0.640252
m_melodicDistanceAverage	float64	878	877			1.062176	11.91445	3.644398	1.299764	2.762402	3.391637	4.188613
m_melodicDistanceStd	float64	878	878			1.198006	9.728562	3.592861	1.301583	2.699617	3.31519	4.204944
m_melodicDurationChange	float64	878	870			0	0.942486	0.27086	0.160719	0.156644	0.229502	0.3589
m_melodicIntervalAverage	float64	878	876			1.062176	8.749216	3.227946	0.929219	2.603918	3.110783	3.704589
m_melodicIntervalStd	float64	878	878			1.198006	4.91417	2.80345	0.600795	2.381227	2.767982	3.148829
m_mostCommonIntervalPresence	float64	878	858			0.121448	0.833333	0.297348	0.088926	0.231043	0.283102	0.352689
m_mostCommonMelodicInterval	object	878	12	M2	501							
m_numberOfMelodicDistances	int64	878	10			2	11	6.427107	1.546997	5	6	8

Appendix C: Set of short music scores used for feature construction testing

1	General 
2	Tie 
3	Ornaments 
4	Articulations 
5	Dynamics 
6	Rests 
7	Tuplets 
8	Time signature changes 
9	Key signature changes 
10	Accidentals 

11	Chords  C Cm C+ Cdim Csus Cadd4 C7 Cm7 CMaj7
12	Voices 
13	Repetitions 
14	Parts 
15	All durations 

Appendix D: Python implementation

Feature construction script

```
# -*- coding: utf-8 -*-
import pandas as pd
import music21 as ms

def createNotesDataframe(c):
  notesList = []

  curKey = ''
  scale = None
  noChecks = False

  for n in c.flat.notes:
    if n.duration.isGrace is False and n.quarterLength != 0:

      scale, curKey, noChecks = getScales(c, n, curKey, scale, noChecks)

      for p in n.pitches:
        info = {
          'durationQL': float(n.quarterLength),
          'durationFullName': n.duration.fullName,
          'pitchClass': p.pitchClass,
          'pitchNumber': p.ps,
          'pitchName': p.name,
          'measure': n.measureNumber,
          'pitchAccidental': True
        }

        if p.pitchClass in scale:
          info['pitchAccidental'] = False

        notesList.append(info)

  df = pd.DataFrame(notesList)

  return df


def createMelodyDataframe(c):
  melodyList = []
  s = c.voicesToParts()

  for p in s.parts:
    listNotes = p.flat.notesAndRests
    for i in range(1, len(listNotes)):
      curNote = listNotes[i - 1]
      nextNote = listNotes[i]

      if curNote.isRest or nextNote.isRest or curNote.quarterLength == 0:
        continue

      info = {
        'Distance': nextNote.pitches[-1].ps - curNote.pitches[-1].ps,
        'RhythmChange': curNote.duration.quarterLength != nextNote.duration.quarterLength
      }
      info['Move'] = 'Ascending' if info['Distance'] > 0 else 'Descending' if info['Distance']
< 0
                                                                              else  'Unison'
      info['Interval'] = abs(info['Distance']) % 12

      if (info['Interval'] == 0) & (info['Distance'] != 0):
        info['Interval'] = 12

      melodyList.append(info)

  df = pd.DataFrame(melodyList)
  return df
```

```python
def getScales(c, n, curKey, scale, noChecks):
  ks = n.getContextByClass('KeySignature')

  if (noChecks == False):
    if (ks is None) & (scale is None):
      ks = c.analyze('key')
      scale = [sc.pitchClass for sc in ks.getScale().pitches]
      noChecks = True
    else:
      if (curKey != ks) | (scale is None):
        scale = [sc.pitchClass for sc in ks.getScale().pitches]
        curKey = ks

  return scale, curKey, noChecks

def range1(end): return range(1, end + 1)

def getHarmonyDataframe(sChords):
  chordList = []
  intervalos = []
  partialChords = 0

  allChords = sChords.flat.getElementsByClass('Chord')

  # create pandas dataframe of chords information
  for i in allChords:

    if i.pitchClassCardinality == 2: partialChords += 1

    # only account for chords of 3 or more notes
    if i.pitchClassCardinality > 2:
      acorde = {
        'name': i.commonName,
        'quarterLength': i.quarterLength,
        'numberOfNotesInAChord': i.multisetCardinality,
        'chordWithSamePitchClass': i.multisetCardinality != i.pitchClassCardinality,
        'numberOfClassesInAChord': i.pitchClassCardinality,
        'notesInChord': ",".join([c.nameWithOctave for c in i.pitches]),
        'pitchClassesString': i.orderedPitchClassesString,
        'chordType': getChordType(i),
        'measure': i.measureNumber
      }

      chordList.append(acorde)

    # gather all the intervals between all pairs of notes in the chord
    for p1 in range(0, len(i.pitches) - 1):
      for p2 in range(p1 + 1, len(i.pitches)):
        distance = abs(i.pitches[p2].ps - i.pitches[p1].ps) % 12
        distance = int(distance) if distance > 0 else 12
        intervalos.append(distance)

  df = pd.DataFrame(chordList)
  chordInfo = {'dfChords': df, 'intervalList': intervalos, 'partialChords': partialChords}

  return chordInfo


def getKeySignatureChanges(s):
  curKey = ''
  keyChanges = 0
  # get key signature changes
  for key in s.parts[0].recurse(classFilter='KeySignature'):
    if curKey != key.sharps:
      keyChanges += 1
    curKey = key.sharps

  return keyChanges - 1 if keyChanges > 0 else 0


def getTimeSignatureInfo(s):
  ts = s.parts[0].flat.getElementsByClass('TimeSignature')
  timeInfo = {
    'timeChanges': 0, 'numerator': 0,
    'denominator': 0, 'ratio': '' }

  if len(ts) >= 1:
    timeInfo['numerator'] = ts[0].numerator
    timeInfo['denominator'] = ts[0].denominator
    timeInfo['ratio'] = ts[0].ratioString
    if len(ts) > 1:
      cur_time = ''
      for tim in ts:
        if cur_time != tim.ratioString:
          timeInfo['timeChanges'] += 1
        cur_time = tim.ratioString

  return timeInfo


def getChordType(notes):
  # representation of pitch class in a chord
  chordTypes = {
    '0,3,7': 'Minor Triad',
    '0,4,7': 'Major Triad',
    '0,3,6': 'Diminished Triad',
    '0,4,8': 'Augmented Triad',
    '0,1,5,8': 'Major-Seventh',
    '0,3,5,8': 'Minor-Seventh',
    '0,3,6,8': 'Dominant-Seventh'
  }

  normalOrder = notes.normalOrder
  root = normalOrder[0]
  normalForm = ",".join([str((pc - root) % 12) for pc in normalOrder])

  if normalForm in chordTypes:
    return chordTypes[normalForm]
  elif len(normalOrder) == 3:
    return 'Other Triad'
  elif len(normalOrder) == 4:
    return 'Other4Chord'
  elif len(normalOrder) > 4:
    return 'Complex Chord'


def getRhythmFeatures(s, df):
  dfCounts = df['durationQL'].value_counts(normalize=True)
  dfCountDurationNames = df['durationFullName'].value_counts(normalize=True)

  info = {
    'r_minimumDuration': df['durationQL'].min(),
    'r_maximumDuration': df['durationQL'].max(),
    'r_noteDurationAverage': df['durationQL'].mean(),  # average of all the durations
    'r_noteDurationStdDev': df['durationQL'].std(),  # standard deviation of all durations
    'r_noteDensity': len(df) / float(s.duration.quarterLength),
    'r_mostCommonNotePresence': dfCounts.iloc[0],  # fraction of number of notes with the most
frequent duration
    'r_numberOfDistinctDurations': len(dfCounts),
    'r_twoMostCommonNotePresence': dfCounts.iloc[0],
    'r_distanceTwoMostFrequent': 0, # default value is 0
    'b_numberOfNotes': len(df),
    'b_scoreLengthInQuarterNotes': float(s.duration.quarterLength)
  }

  info['r_mostCommonNoteDuration'] = dfCounts.index[0]

  # time signature info
  timeInfo = getTimeSignatureInfo(s)
  info['r_timeSignatureChanges'] = timeInfo['timeChanges']
  info['r_tripleMeter'] = timeInfo['numerator'] in (3, 9)
  info['r_dupleMeter'] = timeInfo['ratio'] in ('2/2', '2/4', '4/4', '6/8')

  if info['r_numberOfDistinctDurations'] > 1:
    info['r_twoMostCommonNotePresence'] = dfCounts.iloc[0] + dfCounts.iloc[1]
    durations = sorted([dfCounts.index[0], dfCounts.index[1]], reverse=True)
    info['r_distanceTwoMostFrequent'] = durations[0] / durations[1]

  info['r_durationRange'] = info['r_minimumDuration'] / info['r_maximumDuration']

  # duration histogram
  histogramKeys = ['Breve', 'Whole', 'Half', 'Quarter', 'Eighth', '16th', '32nd', 'Dotted 
Whole', 'Dotted Half', 'Dotted Quarter', 'Dotted Eighth', 'Dotted 16th', 'Dotted 32nd']

  totalNonBasic = 1
  for key in histogramKeys:
    keyUsed = 'r_dh_' + key.replace(' ', '_')
    info[keyUsed] = 0
    if key in dfCountDurationNames.index.values:
      info[keyUsed] = dfCountDurationNames.loc[key]
      totalNonBasic -= info[keyUsed]

  info['r_nonBasicDurationsPresence'] = totalNonBasic if totalNonBasic > 0 else 0

  # add grace note presence
  graceNotes = 0
  expressions = 0
  for i in s.flat.notes:
    if i.quarterLength == 0:
      graceNotes += 1
    expressions += len(i.expressions)
  info['r_graceNotePresence'] = graceNotes / len(df)
  info['r_expressionsPresence'] = expressions / len(df)

  # add rest information
  notesAndRests = s.flat.notesAndRests
  total = len(notesAndRests)
  rests = [i.duration.quarterLength for i in notesAndRests if i.isRest]
  info['r_restPresence'] = len(rests) / total

  return info


def getPitchFeatures(s, df):
  dfClasses = df['pitchClass'].value_counts(normalize=True)
  dfPitches = df['pitchNumber'].value_counts(normalize=True)

  distance = abs(dfClasses.index[0] - dfClasses.index[1])

  infoPitch = {
    'p_minimumPitch': df['pitchNumber'].min(),
    'p_maximumPitch': df['pitchNumber'].max(),
    'p_pitchAverage': df['pitchNumber'].mean(),
    'p_pitchStd': df['pitchNumber'].std(),
    'p_numberOfPitchClasses': df['pitchClass'].nunique(),
    'p_numberOfPitches': df['pitchNumber'].nunique(),
    'p_distanceStrongestPitches': abs(dfPitches.index[0] - dfPitches.index[1]),
    'p_distanceStrongestPitchClasses': distance if distance < 7 else 12 - distance,
    'p_keySignatureChanges': getKeySignatureChanges(s),
    'p_accidentals': df['pitchAccidental'].mean(),
    'p_mostCommonPitchPresence': dfPitches.iloc[0],
    'p_strengthOfTopPitches': dfPitches.iloc[1] / dfPitches.iloc[0] if (len(dfPitches) > 1) 
else 0
  }

  infoPitch['p_pitchRange'] = infoPitch['p_maximumPitch'] - infoPitch['p_minimumPitch']

  for i in range(0, 12): infoPitch['p_ph' + f'{i+1:02}'] = 0
  for i in range(0, len(dfClasses)): infoPitch['p_ph' + f'{i+1:02}'] = dfClasses.iloc[i]

  return infoPitch


def getHarmonyFeatures(sChords, params):
  info = {}

  df = params['dfChords']
  intervalos = params['intervalList']
  # chord features
  if len(df) > 0:
    dfChords = df['pitchClassesString'].value_counts(normalize=True)
    infoChords = {
      'h_chordNotesAverage': df['numberOfNotesInAChord'].mean(),
      'h_chordNotesStd': df['numberOfNotesInAChord'].std(),
      'h_chordPitchClassesAverage': df['numberOfClassesInAChord'].mean(),
      'h_chordPitchClassesStd': df['numberOfClassesInAChord'].std(),
      'h_chordsWithOctave': df['chordWithSamePitchClass'].mean(),
      'h_uniqueChords': df['notesInChord'].nunique() / len(df),
      'h_uniqueChordDurations': df['quarterLength'].nunique(),
      'h_chordDurationAverage': df['quarterLength'].mean(),
      'h_chordDensity': len(df) / float(sChords.duration.quarterLength),
      'h_mostCommonChordSet': dfChords.iloc[0],
      'b_numberOfChords': len(df)
    }

    info.update(infoChords)

    dfChordTypes = df['chordType'].value_counts(normalize=True)
    chordTypes = {
      'h_chordType_01_Minor_Triad': dfChordTypes.get('Minor Triad', 0),
      'h_chordType_02_Major_Triad': dfChordTypes.get('Major Triad', 0),
      'h_chordType_03_Diminished_Triad': dfChordTypes.get('Diminished Triad', 0),
      'h_chordType_04_Augmented_Triad': dfChordTypes.get('Augmented Triad', 0),
      'h_chordType_05_OtherTriads': dfChordTypes.get('Other Triad', 0),
      'h_chordType_06_MinorSeventh': dfChordTypes.get('Minor-Seventh', 0),
      'h_chordType_07_DominantSeventh': dfChordTypes.get('Dominant-Seventh', 0),
      'h_chordType_08_MajorSeventh': dfChordTypes.get('Major-Seventh', 0),
      'h_chordType_09_Other4Chords': dfChordTypes.get('Other4Chord', 0),
      'h_chordType_10_ComplexChords': dfChordTypes.get('Complex Chord', 0)
    }

    info.update(chordTypes)

  # harmonic intervals features
  if len(intervalos) > 0:

    vIntervals = {1: 'm2', 2: 'M2', 3: 'm3', 4: 'M3', 5: 'P4', 6: 'TT',
                  7: 'P5', 8: 'm6', 9: 'M6', 10: 'm7', 11: 'M7', 12: 'P8'}

    # create a pandas series with the vertical intervals, reindex to have a 1 to 12 series
(fill with 0 the missing values)
    intervalCount = pd.Series(intervalos).value_counts(normalize=True).reindex(range1(12),
fill_value=0)

    # add intervals count
    for key, value in intervalCount.iteritems():
      info['h_harmonic_Interval_' + f'{key:02}_' + vIntervals[key]] = intervalCount[key]

    info['h_topVerticalInterval'] = vIntervals[intervalCount.argmax()]
    info['h_topVerticalIntervalPresence'] = intervalCount.max()

  info['h_partialChords'] = params['partialChords'] / float(sChords.duration.quarterLength)
  return info


def getMelodicFeatures(s, df):
  info = {}
  dfIntervals = df['Interval'].value_counts(normalize=True)

  melodyIntervals = {
    'm_intervals_00_unison': dfIntervals.get(0, 0),
    'm_intervals_01_minorSecond': dfIntervals.get(1, 0),
    'm_intervals_02_majorSecond': dfIntervals.get(2, 0),
    'm_intervals_03_minorThird': dfIntervals.get(3, 0),
    'm_intervals_04_majorThird': dfIntervals.get(4, 0),
    'm_intervals_05_perfectFourth': dfIntervals.get(5, 0),
    'm_intervals_06_Tritone': dfIntervals.get(6, 0),
    'm_intervals_07_perfectFifth': dfIntervals.get(7, 0),
    'm_intervals_08_minorSixth': dfIntervals.get(8, 0),
    'm_intervals_09_majorSixth': dfIntervals.get(9, 0),
    'm_intervals_10_minorSeventh': dfIntervals.get(10, 0),
    'm_intervals_11_majorSeventh': dfIntervals.get(11, 0),
    'm_intervals_12_Octave': dfIntervals.get(12, 0)
  }
  info.update(melodyIntervals)

  mIntervals = {0: 'Unison', 1: 'm2', 2: 'M2', 3: 'm3', 4: 'M3', 5: 'P4', 6: 'TT',
                7: 'P5', 8: 'm6', 9: 'M6', 10: 'm7', 11: 'M7', 12: 'P8'}

  info['m_mostCommonMelodicInterval'] = mIntervals[dfIntervals.index[0]]
  info['m_mostCommonIntervalPresence'] = dfIntervals.iloc[0]
  info['m_largeIntervals'] = len(df[df['Distance'] > 12]) / len(df)

  info['m_melodicIntervalAverage'] = df['Interval'].mean()
  info['m_melodicIntervalStd'] = df['Interval'].std()
  info['m_melodicDistanceAverage'] = abs(df['Distance']).mean()
  info['m_melodicDistanceStd'] = abs(df['Distance']).std()

  dfDistance = df['Distance'].value_counts(normalize=True)
  info['m_numberOfMelodicDistances'] = len(dfDistance[dfDistance >= 0.05])

  dfMoves = df['Move'].value_counts(normalize=True)
  melodyMove = {
    'm_directionUp': dfMoves.get('Ascending', 0),
    'm_directionDown': dfMoves.get('Descending', 0),
    'm_directionSame': dfMoves.get('Unison', 0)
  }
  info.update(melodyMove)

  info['m_melodicDurationChange'] = df['RhythmChange'].mean()

  contourChanges = 0
  contours = df['Move'].tolist()
  for i in range(1, len(contours)):
    if contours[i - 1] != contours[i]:
      contourChanges += 1
  info['m_melodicContourChange'] = contourChanges / len(contours)

  return info


def getFeatures(row):
  c = ms.converter.parseFile(row['filename'])

  df = createNotesDataframe(c)
  # rhythm features
  fs = getRhythmFeatures(c, df)
  for key in fs:  row[key] = fs[key]  ## copy features to row
  # pitch features
  fs = getPitchFeatures(c, df)
  for key in fs:  row[key] = fs[key]  ## copy features to row
  # melody features
  dfMelodic = createMelodyDataframe(c)
  fs = getMelodicFeatures(c, dfMelodic)
  for key in fs:  row[key] = fs[key]  ## copy features to row
  # harmony features
  chordInfo = getHarmonyDataframe(c.chordify())
  fs = getHarmonyFeatures(c, chordInfo)
  for key in fs:   row[key] = fs[key]  ## copy features to row

  return row


# open dataset of music scores and paths
dfDataSet = pd.read_csv('xls/output/FinalDataset.csv', sep=';', usecols=['composer', 'title',
'period', 'filename'],
                        encoding='utf-8-sig')
# construct features
df = dfDataSet.apply(getFeatures, axis=1)
# export original dataset with the features added
df.to_csv('xls/output/scoresDataSet.csv', index=False, sep=';', columns=df.columns,
encoding='utf-8-sig')
```

Classifier model script

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
from sklearn.metrics import precision_recall_fscore_support
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold,
cross_val_predict, GridSearchCV
import pprint as pp


def exportResults(df, t_target, pred, metrics, dfAccuracy):
  cols = ['ID', 'composer', 'title', 'Class', 'period']
  dfResults = df
  dfResults['Class'] = t_target
  for k in pred:
    dfResults['Predicted_' + k] = pred[k]
    dfResults['Correct_' + k] = np.where(dfResults.Class == dfResults['Predicted_' + k], 1, 0)
    cols.append('Predicted_' + k)
    cols.append('Correct_' + k)

  # predictions of each record by feature set
  dfResults.to_excel('xls/Output/models/' + '_ByPieceByFeatureType.xlsx', columns=cols,
index=False)

  dfRecall = pd.DataFrame({'period': periodColumns})
  for i in metrics:  dfRecall[i] = metrics[i]['recall']
  dfRecall.T.to_excel('xls/Output/models/recall_' + '.xlsx', index=True)

  dfPrecision = pd.DataFrame({'period': periodColumns})
  for i in metrics:  dfPrecision[i] = metrics[i]['precision']
  dfPrecision.T.to_excel('xls/Output/models/precision_' + '.xlsx', index=True)

  dfFScore = pd.DataFrame({'period': periodColumns})
  for i in metrics:  dfFScore[i] = metrics[i]['fscore']
  dfFScore.T.to_excel('xls/Output/models/fscore' + '.xlsx', index=True)

  dfSupport = pd.DataFrame({'period': periodColumns})
  for i in metrics:  dfSupport[i] = metrics[i]['support']
  dfSupport.T.to_excel('xls/Output/models/support' + '.xlsx', index=True)

  dfAccuracy.to_excel('xls/Output/models/accuracy' + '.xlsx', index=True)


def dataPreprocessing(dfDataSet):
  # fill missing values with 0 (chordtypes info)
  dfDataSet.fillna(0, inplace=True)

  # derived features
  dfDataSet['r_timeSignatureChanges'] = dfDataSet['r_timeSignatureChanges'] == 0
  dfDataSet['r_durationRange'] = dfDataSet['r_maximumDuration'] -
dfDataSet['r_minimumDuration']

  dfDataSet['p_pitchRange'] = dfDataSet['p_maximumPitch'] - dfDataSet['p_minimumPitch']
  dfDataSet['p_keySignatureChanges'] = dfDataSet['p_keySignatureChanges'] > 0
  # convert the numeric to categorical
  dfDataSet['p_distanceStrongestPitchClasses'] =
dfDataSet['p_distanceStrongestPitchClasses'].astype(str)

  # create list of categorical and numeric variables
  arrContinuous = []
  arrCategorical = []
  for i in dfDataSet.columns:
    if i[1] == '_':
      colType = dfDataSet[i].dtypes.name
      if colType in ['float64', 'int64']:
        arrContinuous.append(i)
      elif colType in ['object']:
        arrCategorical.append(i)
  ######## CLAMP TRANSFORMATION
  iqr = dfDataSet[arrContinuous].quantile(0.75) - dfDataSet[arrContinuous].quantile(0.25)
  high = dfDataSet[arrContinuous].quantile(0.75) + (1.5 * iqr)
  low = dfDataSet[arrContinuous].quantile(0.25) - (1.5 * iqr)
  outliers_high = (dfDataSet[arrContinuous] > high)
  outliers_low = (dfDataSet[arrContinuous] < low)
  dfDataSet[arrContinuous] = dfDataSet[arrContinuous].mask(outliers_high, high, axis=1)
  dfDataSet[arrContinuous] = dfDataSet[arrContinuous].mask(outliers_low, low, axis=1)

  ### NORMALISATION
  transformation = preprocessing.MinMaxScaler(feature_range=(0, 1))
  dfDataSet[arrContinuous] = transformation.fit_transform(dfDataSet[arrContinuous])

  # Convert categorical to binary
  for col in arrCategorical:
    print(col)
    for i in dfDataSet[col].unique():
      # create a new column with field name + '_' + level name with 0 or 1 values
      dfDataSet[col + '_' + i] = (dfDataSet[col] == i)
    dfDataSet.drop(col, axis=1, inplace=True)

  return dfDataSet


def getBestFeaturesUsingRandomTree(X, y):
  model = RandomForestClassifier(max_depth=40, n_estimators=1000, criterion='gini',
bootstrap=False, random_state=42)
  model.fit(X, y)
  featureImportances = model.feature_importances_
  importantFeatures = sorted(zip(X.columns, featureImportances), key=lambda x: x[1],
reverse=True)
  dfFeat = pd.DataFrame({'Variable': X.columns, 'Importance': featureImportances})
  dfFeat = dfFeat.sort_values(by='Importance', ascending=False)
  return dfFeat['Variable'].tolist(), dfFeat


df = pd.read_csv('xls/output/scoresDataSet.csv', sep=';', encoding='utf-8-sig')
df = df.reset_index(drop=True)
# drop metadata information
dfDataSet = df.drop(['ID', 'composer', 'title', 'filename'], axis=1)
# data preparation
dfDataSet = dataPreprocessing(dfDataSet)

periodColumns = ['Baroque', 'Classical', 'Romantic', 'Modern']

# feature types list
features = [
  ('Rhythm', [col for col in dfDataSet.columns if col.startswith('r_')]),
  ('Pitch', [col for col in dfDataSet.columns if col.startswith('p_')]),
  ('Harmony', [col for col in dfDataSet.columns if col.startswith('h_')]),
  ('Melody', [col for col in dfDataSet.columns if col.startswith('m_')]),
  ('All', [col for col in dfDataSet.columns if col != 'period'])
]

featureCategories = [ft for ft, col in features]

# features dataset
t_features = dfDataSet
# labels dataset
t_target = dfDataSet.pop('period')

# radial kernel SVM
clf = svm.SVC(kernel='rbf', C=100, random_state=0, gamma='auto'),  # SVC with radial kernel

pred = {}
metrics = {}
accuracy = []
results = {}

# create a model per group of features
for featureType, cols in features:

  # select the features for a group
  X = t_features[cols]
  y = t_target
  # 10-fold cross validation
  skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=52)

  # PARAMETER TUNING
  # param_grid = [
  #     {   'C': [0.1, 1, 10, 100],
  #         'gamma': [0.01, 0.05, 0.1, 0.3, 0.7, 1,'auto'],
  #     }
  # ]
  # grid = GridSearchCV(clf, cv=skf, param_grid=param_grid)
  # grid.fit(X, y)  #
  # pp.pprint(grid.best_estimator_)
  # pp.pprint(grid.best_params_)
  # pp.pprint(grid.cv_results_['mean_test_score'])
  # # pp.pprint(grid.param_grid)
  #

  # random forest variable ranking
  feat, dfFeat, tree1 = getBestFeaturesUsingRandomTree(X, y)
  plt.figure(figsize=(20, 10))
  sns.factorplot(x="Importance", y="Variable", data=dfFeat[:5], kind="bar",
                 size=3, aspect=1.9, color='royalblue')
  plt.ylabel('')
  plt.tight_layout()
  plt.title("Variable Ranking / " + featureType + " features", fontsize=13,
horizontalalignment='center')
  plt.savefig('xls/Output/models/VarRanking_' + featureType + '.png')
  dfFeat.to_excel('xls/Output/models/VarRanking_' + featureType + '.xlsx', index=True)

  # train and get predictions of the model
  predictions = cross_val_predict(clf, X, y, cv=skf)
  scores = y == predictions
  print('{0:10}({1:3})\tAccuracy:{2:0.5f}'.format(featureType, len(cols), scores.mean()))

  # get evaluation metrics
  precision, recall, fscore, support = precision_recall_fscore_support(y, predictions,
average=None,
                                                                       labels=periodColumns,
warn_for=('F-score'))
  dfClass = precision_recall_fscore_support(y, predictions, labels=periodColumns, warn_for=('F-
score'))

  data = {'period': periodColumns,
          'precision': precision,
          'recall': recall,
          'fscore': fscore,
          'support': support}

  accuracy.append(scores.mean())

  metrics[featureType] = pd.DataFrame(data, columns=['period', 'precision', 'recall', 'fscore',
'support'])
  pred[featureType] = predictions

  confMatrix = pd.crosstab(y, predictions).reindex_axis(periodColumns,
               axis=1).reindex_axis(periodColumns, axis=0)
  confMatrix.fillna('0', inplace=True)

  confMatrix.to_excel('xls/Output/models/ConfMatrix_' + featureType + '.xlsx')

  # confusion matrix with %
  confMatrix2 = confMatrix.astype(np.float) / confMatrix.sum(axis=1)
  colsum = confMatrix.sum(axis=1)
  confMatrix2 = confMatrix
  for index, row in confMatrix2.iterrows():
    for per in periodColumns:
      confMatrix2.loc[index, per] = row[per] / colsum[index]

  confMatrix2.to_excel('xls/Output/models/ConfMatrixPercent_' + featureType + '.xlsx')

dfAccuracy = pd.DataFrame(data={'Accuracy': accuracy}, index=featureCategories)
exportResults(clf, df, t_target, pred, metrics, dfAccuracy)
```

Summary statistics

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm, tqdm_pandas
import seaborn as sns

def exportStats(features, df):
  # loop each feature category
  for type, cols in features:
    stats = []
    # loop each field/attribute
    for col in cols:
      info = {}
      ser = df[col]
      colType = ser.dtypes.name
      # get statistics of the column
      info['field'] = col
      info['count'] = ser.count()
      info['type'] = ser.dtype.name
      info['unique'] = len(ser.unique())
      #
      if colType in ['float64', 'int64']:
        info['min'] = ser.min()
        info['max'] = ser.max()
        info['mean'] = ser.mean()
        info['std'] = ser.std()
        info['25'] = ser.quantile(.25)
        info['50'] = ser.quantile(.50)
        info['75'] = ser.quantile(.75)
      elif colType in ['object']:
        info['top'] = ser.value_counts().index[0]
        info['freq'] = ser.value_counts().iloc[0]

      stats.append(info)

    dfStats = pd.DataFrame(stats,columns=['field', 'type', 'count', 'unique', 'top',
                      'freq', 'min', 'max', 'mean', 'std','25', '50', '75'])
    dfStats.to_excel('xls/DataStats/' + type + '_Features.xlsx', index=False)


def createBars(dfGroup):
  for i in tqdm(dfGroup.columns):
    dfGroup[i].plot.bar(title=i, edgecolor='k')
    plt.xlabel('')
    plt.ylabel('Avg values')
    plt.tight_layout()
    plt.axes().yaxis.grid(color='grey', linestyle='dashed')
    plt.axes().set_axisbelow(True)
    plt.savefig('xls/DataStats/bars/' + i + '.png')
    plt.clf()


def createBoxPlotsPerPeriod(dfDataSet):
  for i in tqdm(dfGroup.columns):
    sns.factorplot(kind='box', x="period", y=i, data=dfDataSet,
               order=periodColumns, width=0.6, fliersize=0.5, aspect=1.1, size=4)
    plt.tick_params(labelsize=14)
    plt.title(i, fontsize=18)
    plt.xlabel('')
    plt.ylabel('')
    plt.tight_layout()
    plt.savefig('xls/DataStats/boxPlots/PerPeriod/' + i + '.png')
    plt.clf()


def createBoxPlots(dfDataSet):
  for i in tqdm(dfGroup.columns):
    sns.boxplot(y=dfDataSet[i])
    plt.savefig('xls/DataStats/boxPlots/' + i + '.png')
    plt.clf()
```

```python
def createHistograms(dfDataSet):
  for i in tqdm(dfDataSet.columns):
    if dfDataSet[i].dtype != 'object':
      dfDataSet[i].plot.hist(title=i, color='#5A8CA8', bins=50)
      plt.locator_params(nbins=10)
      plt.savefig('xls/DataStats/hist/' + i + '.png')
      plt.clf()


def normalisationExample():
  cols = ["r_dh_Eighth", "m_melodicIntervalAverage"]
  sns.regplot(y=dfDataSet["r_dh_Eighth"], x=dfDataSet["m_melodicIntervalAverage"],
              fit_reg=False, marker="+", scatter_kws={"s": 20},
              label='Before normalisation', color=None)
  plt.show()
  transformation = preprocessing.MinMaxScaler(feature_range=(0, 1))
  dfDataSet[cols] = transformation.fit_transform(dfDataSet[cols])
  sns.regplot(y=dfDataSet["r_dh_Eighth"], x=dfDataSet["m_melodicIntervalAverage"],
              fit_reg=False, scatter_kws={"color": "darkred", "alpha": 1, "s": 15},
              label='After normalisation')
  leg = plt.legend(loc='upper right', frameon=True, fontsize=13, markerscale=1.8)
  leg.get_frame().set_facecolor('lightgrey')
  plt.show()


def correlations(features, dfDataSet):
  from scipy.stats import pearsonr

  for ftype, cols in features:
    dfD = dfDataSet[cols].dropna()
    cor = dfD.corr(method='pearson')
    # Generate a mask for the upper triangle
    mask = np.zeros_like(cor, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    # Set up the matplotlib figure
    sns.set(font_scale=0.5)
    # f, ax = plt.subplots(figsize=(11, 9))
    # Generate a custom diverging colormap
    cmap = sns.diverging_palette(220, 10, as_cmap=True)
    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(cor, mask=mask, cmap=cmap, vmax=.3, center=0,
                annot=True, annot_kws={"size": 5}, fmt=".2f",
                square=True, linewidths=.5, cbar_kws={"shrink": .5})
    plt.xticks(rotation=90)
    plt.yticks(rotation=0)
    # show correlation heatmap
    plt.show()

    # export correlation matrix to exxcel
    cor.to_excel('xls/DataStats/Correlations/' + ftype + '_Features_Matrix.xlsx', index=True)
    corr = cor.unstack().reset_index()
    corr.columns = ['var1', 'var2', 'coef']
    corr = corr[corr['coef'].abs() >= 0.8]
    corr = corr[corr['coef'].abs() < 1]
    corr['pvalue'] = 0
    for per in periodColumns:  corr[per] = 0
    todelete = []
    for i, row in corr.iterrows():
      if i not in todelete:
        corrtest = pearsonr(dfD[row['var1']], dfD[row['var2']])
        corr.loc[i, 'pvalue'] = corrtest[1]
        # calculate correlation per column
        for per in periodColumns:
          dfClass = dfDataSet[dfDataSet['period'] == per]
          dfClass = dfClass[cols].dropna()
          corrtest = pearsonr(dfClass[row['var1']], dfClass[row['var2']])
          corr.loc[i, per] = corrtest[0]

        for j, row2 in corr.iterrows():
          if (row['var1'] == row2['var2']) & (row['var2'] == row2['var1']):
            todelete.append(j)

    corr.drop(todelete, inplace=True)
    corr['pvalue'] = corr['pvalue'].apply(lambda x: '%.4f' % x)
    corr.to_excel('xls/DataStats/Correlations/' + ftype + '_Features_Pairs.xlsx', index=True)

# MAIN SCRIPT
dfDataSet = pd.read_csv('xls/output/scoresDataset.csv', sep=';')

periodColumns = ['Baroque', 'Classical', 'Romantic', 'Modern']

dfGroup = dfDataSet.groupby('period').mean().reindex(periodColumns)
dfGroup.T.to_excel('xls/DataStats/FeaturesMeanByPeriod.xlsx', columns=periodColumns)

features = [
  ('Rhythm', [col for col in dfDataSet.columns if col.startswith('r_')]),
  ('Pitch', [col for col in dfDataSet.columns if col.startswith('p_')]),
  ('Harmony', [col for col in dfDataSet.columns if col.startswith('h_')]),
  ('Melody', [col for col in dfDataSet.columns if col.startswith('m_')]),
  ('All', [col for col in dfDataSet.columns if col != 'period'])
]

exportStats(features, dfDataSet)
createBoxPlots(dfDataSet)
createBars(dfGroup)
createHistograms(dfDataSet)
correlations(features, dfDataSet)
normalisationExample()
createBoxPlotsPerPeriod(dfDataSet)
```

Assignment of labels

```
import os
import pandas as pd
import music21 as ms
import sys


def getPeriod(comp, df):
  jaro = 0
  auxComposer = ''
  auxPeriod = ''
  auxYears = ''
  if comp != '':
    for composer, period, years in zip(df['composer'], df['period'], df['years']):
      newJaro = fuzz.token_set_ratio(comp, composer)
      if newJaro > jaro:
        auxComposer = composer
        auxPeriod = period
        auxYears = years
        jaro = newJaro
  return auxComposer, auxPeriod, auxYears

def getInfo(row):
  try:
    c = ms.converter.parse(row['filename'])
  except:
    row['error'] = str(sys.exc_info()[1])
    return row

  if c.metadata.composer is not None:
    row['composerOld'] = c.metadata.composer
  else:
    row['composerOld'] = row['paths']

  title = ''
  if c.metadata.parentTitle != 'None':
    title = c.metadata.parentTitle

  if c.metadata.title != 'None':
    if c.metadata.title != '': title = c.metadata.title if title == ''
           else title + ' ' + c.metadata.title

  if c.metadata.movementNumber is not None:
    if c.metadata.movementNumber != '': title = title + ' : ' + c.metadata.movementNumber
  if c.metadata.movementName is not None:
    if not c.metadata.movementName.endswith('.mxl'):
      if c.metadata.movementName != '': title = title + ' : ' + c.metadata.movementName

  if c.metadata.number is not None:
    title = title + ' : ' + c.metadata.number if c.metadata.number != '' else title

  title = title.replace('.mxl', '').replace('_', ' ')

  row['title'] = title
  row['composer'], row['period'], row['years'] = getPeriod(row['composerOld'], dfComposers)

  return row


# MAIN SCRIPT

# open composer lookup table
dfComposers = pd.read_excel('xls/input/composers.xlsx', Sheet='Sheet1', index=False,
encoding='utf-8-sig')

id = 1
# empty lists, ids and score paths
ids = []
scores_path = []
scores_files = []
scores_sources = []

datasetOthersPath = ['scores/kern', 'scores/mxl']
# loop through all files in the folders
for path in datasetOthersPath:
  source = path.replace('scores/', '')
  for root, directories, filenames in os.walk(path):
    for filename in filenames:
      scores_files.append(os.path.join(root, filename))
      ids.append(id)
      scores_path.append(root.replace(path, '').replace('\\', ''))
      scores_sources.append(source)
      id += 1

df = pd.DataFrame()
# add id and score files
df['ID'] = ids
df['filename'] = scores_files
df['paths'] = scores_path
df['source'] = scores_sources

# add columns of composer, period, years
df = df.apply(getInfo, axis=1)

columns = ['composer', 'composerOld', 'title', 'period', 'years',
           'source', 'filename', 'error']

df.to_csv('xls/output/FinalDataset.csv', index=False, sep=';',
          encoding='utf-8-sig',columns=columns)
```

Crawl MuseScore website to gather music scores info

```
from bs4 import BeautifulSoup
import pandas as pd
import os
import time
import requests

headers = {
  "User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"}

payload = {
  "name": "************",
  "pass": "************",
  "form_build_id": "form-hKBAJtBDOaaeZmpsmOa1bc9fSHkTqe4BZbABscHo4kY",
  "form_id": "user_login_form",
```

```python
  "op": "Log+in"
}

base_url = 'https://musescore.com/'
login_page = 'https://musescore.com/user/login'

s = requests.Session()
s.post(login_page, data=payload)
# list of users with classical music scores
dfUsers = pd.read_excel('xls/input/Muse_Score_Users.xlsx', sheetname='Hoja1')

lists = []

for user_page in dfUsers['URL']:
  # scores with only one part
  my_url = user_page + '?parts=1'
  pag = 1
  print(user_page)
  url_list = []

  while True:
    time.sleep(1)
    print('Page:' + str(pag))
    u = s.get(my_url, headers=headers).content
    soup = BeautifulSoup(u, 'html.parser')

    for score in soup.find_all('a', attrs={'rel': 'bookmark'}):
      url_list.append(base_url + score['href'])

    item = soup.find('a', attrs={'rel': 'next'})
    # if there is no Next button, exit loop
    if (item == None): break;

    my_url = user_page + item['href']
    pag += 1

  totUrls = len(url_list)
  cont = 1

  for score_page in url_list:
    print("%s / %s" % (cont, totUrls))
    info = {}
    info['scorePage'] = score_page
    info['userpage'] = user_page
    info['download'] = 'M'

    u = s.get(score_page, headers=headers).content
    soup = BeautifulSoup(u, 'html.parser')

    if soup.title.string == "This score is unavailable | MuseScore":
      print("This score is unavailable | MuseScore")
      continue

    item = soup.find('meta', attrs={'property': 'musescore:composer', 'content': True})

    info['composer'] = item['content'].replace('\n', ' ') if item is not None else ''
    item = soup.find('meta', attrs={'property': 'og:title', 'content': True})
    info['title'] = item['content'].replace('\n', ' ').replace('"', '')
                if item is not None else ''
    # get other metadata info (instrument, pages, measures...)
    xtra = soup.find('div', attrs={'class': 'more-info'})
    trs = xtra.find_all('tr')
    for tr in trs:
      tds = tr.find_all('td')
      if tds[0].text == 'Part names':
        partsTotal = str(tds[1]).replace('<td>', '').replace('<span class="sep">', '').
            replace('</span>', '').replace('</td>', '')
        info[tds[0].text] = "|".join(partsTotal.split('<br/>'))
      else:
        info[tds[0].text] = str(tds[1].text)
    #clean file name
    info['filename'] = info['title'].replace(' ', '_').replace('♯', 'sharp').\
      replace('♭', 'flat').replace('“','-').replace('”', '-').replace('/', '-')
    lists.append(info)
    cont += 1

df = pd.DataFrame(lists)

columns = ['composer','title','filename','download','Pages','Measures','Duration','Key
signature','Part names','Parts','scorePage','userpage']

df.to_excel('xls/output/Muse_Score_Downloads.xlsx', index=False, columns=columns)
```

Download musicXml files from MuseScore

```
from bs4 import BeautifulSoup
import pandas as pd
import os
import requests
import time

headers = {
  "User-Agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
                 (KHTML, like  Gecko) "Chrome/59.0.3071.115 Safari/537.36"}
payload = {
  "name": "*************",
  "pass": "**********",
  "form_build_id": "form-KpVeiZaYjsTnRR5vuwRRhOQ6eyOU3CDB4SgXxtLfuL0",
  "form_id": "user_login_form",
  "op": "Log+in"
}

base_url = 'https://musescore.com/'
login_page = 'https://musescore.com/user/login'
datasetTo = 'scores\\musicxml\\'

s = requests.Session()
s.post(login_page, data=payload)
#open excel file with list of scores to download
dfScores = pd.read_excel('xls/output/Muse_Score_Downloads.xlsx', 'Sheet1')
scores = []
totScores = len(dfScores)

id = 1
for index, row in dfScores.iterrows():
  if row['download'] == 'Y':
    time.sleep(1)
    print("(%s) %s / %s - %s - %s" % (id, index + 1, totScores, row['composer'], row['title']))

    u = s.get(row['scorePage'], headers=headers).content
    # parse HTML to a beautifulsoup object
    soup = BeautifulSoup(u, 'html.parser')
    # find link to MusicXML file
    url = soup.find('a', text='MusicXML')
    # url of musicxml download page
    download_page = base_url + url['href']
    # get url of file to download
    d = s.get(download_page, headers=headers, allow_redirects=False)
    download = d.headers['Location']
    #folder to save the file
    folderTo = datasetTo + row['userpage'].replace('https://musescore.com/', '')
                                          .replace('sheetmusic', '')
    # create folder in case it does not exist
    os.makedirs(folderTo, exist_ok=True)
    #name of the file
    file_nameTo = folderTo + '\\' + row['filename'] + '.mxl'

    with s.get(download, stream=True).raw as response, open(file_nameTo, 'wb') as out_file:
      data = response.read()
      out_file.write(data)
    dfScores.loc[index, 'download'] = 'Done'
    id += 1

dfScores.to_excel('xls/output/Muse_Score_Downloads.xlsx', 'Sheet1', index=False)
```