Western classical composer identification using symbolic data

Francesc Capó Clar

TESI DE MÀSTER UPF / ANY 2015

DIRECTORS DE LA TESI
Joan Serrà & Perfecto Herrera
DEPARTAMENT
DTIC, Music Technology Group

Acknowledgements

First of all, I would like to thank my supervisors, Joan Serrà and Perfecto Herrera for their help, comprehension and for being patient with me during my learning and work.

In particular, I would like to thank my parents and my family, for their unconditional support and love and for helping me in the hardest moments.

Further, an acknowledgement to my all lifelong friends to encouraging me in every moment, and to my colleagues of the SMC master to become an important part of this course. Especially, thanks to my friend Sebastià V. Amengual for fruitful technical discussions and endless encouragement.

Finally, thanks to all the professors from the SCM master to teach, advise and help me during this whole course, as well as the University Pompeu Fabra.

Xesc

Abstract

In this thesis, we address the problem of automatic composer identification using information from score-like representations. This task calls for computing features that capture both specificities of the common practice in each composer's period plus individual specificities with regard to notes, chords, durations, etc. Our goal is to find which are the singular characteristics of each composer and therefore be able to identify them. This information can be useful for musicians as well, in order to study which are the common characteristics between composers and their relationship to the historical period. The particularity of this study is related to the large number of composers used in it (106 composers) and the large of works used as well (9876 works).

With a view to achieve this, we have designed features based on music theory, in order to extract music properties from the scores. We have performed classification experiments using the information from these features using Random Forest and SVM classifiers. Using the classification results from theses experiments, we have used feature selection in order to achieve better accuracies combining information from different music concepts.

Using the best combinations, we have achieved accuracies up to 46,7% (17 composers), 39,0% (47 composers) and 27,6% (106 composers). In addition, we present confusion matrices of the classifications and scalability graphs in terms of accuracy of our approach, fixing the number of composers and the number of works as well. Finally, we analyse the results and we propose further work related to the extension of this study, changing parameters of the features.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Motivation

Music is an art where composers need to use their imagination, inspiration and talent to create works in order to transmit their personal feelings and emotions, in an original and unique way. However, as it happens in other arts, most prominent composers in history conceive musical characteristic traits during their career, which are largely present in all their works.

An experienced listener is able to properly identify a well-known composer because his music *sounds like* his works. This is because our brain is able to find several patterns or note relationships that are repeated in all the works of the same composer. Moreover, even some good musicians are able to create new music, but copying the essence of these other well-known composers. The problem is that we don't always know which are the explicit features that give us a proper feeling of the composer. We suspect that we should extract them from the musical scores.

When a musician plays a work from a composer, he tries to express and transmit to the listeners, the essence of the piece and the composer. This study could give us information about unique composer characteristics placed in their works. Musical performers may use this in order to improve or redefine their interpretations.

The authorship of some works is a topic that has been discussed for several works. One example of this is the authorship of the *Toy symphony*, attributed in the first instance to J. Haydn and then to L. Mozart. Some studies also attribute it to a Benedictine monk called Edmund Angerer. Could we use composer identification mechanisms in order to help to the musicologists to resolve these authorship problems?

The music semantic analysis is a method used by musicologist in order to identify the composer of a work. This method consists of split the work into smaller

parts; then, split these parts in smaller parts as well. This step is repeated several times until we have the music work segmented with excerpts that have music sense. With this study we want to avoid this method, due to the impossibility of perform this task automatically with the scores, but we will try to use alternative methods in order to study the relationship between these notes.

The questions that we aim to answer in this project are: Can we identify the composer of a work? Which are the specific musical elements that lead us to identify the composer? Are these characteristics unique for each one? Is it possible to find features from different composers in one single work? Are these characteristics related to the style or musical period?

1.2 Goal

In this study, we want to extract features from the scores that give us information about the composer of these works. In order to do that, we will use a data set composed of works by western classical composers. We will create several feature set extractors, which will give us information about different musical and statistical aspects from the score notes.

The complexity of the task that we will study lies in the difficulty to solve this problem with a large number of composers (around 100) and a large number of heterogeneous works (around 10.000). Moreover this task has not been performing with this high number of classifying complexities. We also want to study how the number of composers and the number of works impacts on the classification results as well.

1.3 Thesis outline

After this brief introduction, we will focus on the related work in this field in Chapter 2. In Chapter 3 we will explain the methodology used in this study, including the experimental set-up and the features design. The Chapter 4 includes all the results of our study regarding to composer classification, as well as the analysis of the designed features in these classifications and comments about their combinations. Finally, in Chapter 5 we expose the conclusions of this project and we propose some future work that could be done in order to extend and improve the results.

Chapter 2

RELATED WORK

This section comprises a literature overview of the problem of automatic identification of music composers, mostly working on the so-called "Standard Western tradition" during the period comprised between centuries XVII and XX. We will first summarize a number of contributions studying audio data, because we could get ideas from the performance of that task. Then, we will expose studies of symbolic data, which is the type of data that we will use in our experiments.

We think that the essence of each composer is found in some specific statistic measures that can be extracted from their scores. We want to prove that a well trained system, with the correct descriptors and features, should be able to perform the same task successfully as well. The most difficult point is to find which are these descriptors that would give us the information that we need. We want to find any kind of statistical relationship between the notes and their combination that give us some clues to identify the correct composer.

In this chapter, we will introduce the concept of music composer identification, as well as their possible usages. The task that we want to study has two main approaches that we will introduce in this chapter as well: the audio usage (recorded interpretation of the composition) versus the symbolic approach (digitalized score information). Then we will explain which are the basics of the symbolic music and which are the differences between audio-based and symbolic-based approaches. Finally we will talk about the contributions in the field.

2.1 Music Composer Identification

Art creator identification is a problem studied by several disciplines such as painting and music. In these cases, the goal is to find which is the artist/composer of a work, which belong to a group of possible artists. In this study, we will focus in the subclass of music composer identification, taking profit of some of the ideas

and mechanisms that have been developed and used in these sort of problems.

Music composer identification, also known as *MCI*, is one of the tasks that belongs to the *Music Information Retrieval* (MIR) field, as well as other well known problems like genre identification, music recommendation systems or automatic categorization, among others. Like in the most of the MIR tasks, the goal of MCI is to choose which is the most probable artist or composer of a piece among a list of artists/composers which are known by the system due to a previous training or construction of a model.

There is an interest related to MCI that is to find the essence of each composer: a musical unique fingerprint of the artist and present in the majority of his works. This knowledge can be useful for musicians in order to use this information during their rehearsals and performances in order to transmit the soul of each composer. Knowing the differences and the similarities between these composers can be useful for the musicians community and for the studies of the musicologists.

This task has been done regularly by musicologists. Because they tend to perform this task manually, their analysis for the classification is more extensive, analysing the score with musical passages, chord patterns and specific characteristics for these studies. For instance, Harding [1] performs an extensive analysis of Beethoven sonatas of 80 pages, where he exposes their musical structure, meaning, chord progressions, etc. However, in these studies, they don't use the computational tools that we will use in this study. These mechanisms will give us the opportunity to perform this task with a large number of works, and using a systematic and quantitative procedure.

On another hand, this task is performed mostly for classical music because nowadays, most of the popular music that is being produced and recorded, normally they are pretty well documented in their tags, or at least, the people are able to identify them using some of the available fingerprint databases. However, for classical music, some people find old recordings or scores without the composer/artist information. The common approach to identify them is done by musicians and musicologists, which have a strong musical background if this field. They retrieve information from the score, such as note durations, motif recognition, chord progressions or musical interval analysis, among others, which mostly define styles and composers.

2.2 Audio waveform approach

This system tries to classify works by artist using recorded audio from performances of musicians. Although the classification procedure is the same than in all classification tasks, the descriptors used in this way are totally different than the ones used for the symbolic approach.

In the thesis by Melidis [2] the author performs Artist Identification from Electronic Music using the audio approach. The descriptors used in this task include the *MirToolBox* [3] (including typical classical audio descriptors like MFCC's, Spectral Centroid, Zero crossing rate, etc.), *Essentia* [4] descriptors and *Short Time Timbral Codings based on ZIPF's law* [5]. The best results are achieved using *Essentia* descriptors, where he achieves an accuracy of 25% classifying over 86 artists and a 39% of accuracy classifying over 20 artists.

Related to music composer identification of classical music, this task can be done using the recorded audio from a work as well. Actually, MIREX [1] campaigns have evaluated this task since several years ago by the name of *Audio Classical Composer Identification* with the parallel task of *Audio Artist Identification*.

In most of the cases, as MCI is not a hot topic, MIREX submissions for MCI task tend to be generic music classification systems, although they are delivered to the MCI task as well, in order to see how these systems works for this topic. Because of that, the features extracted in these cases are not based on musical theory, but are based on generic descriptors as timbre or rhythm.

The best submitted algorithm for that task in the MIREX edition 2014 achieved an accuracy of 68,76% using a data set of 11 classical composers and 252 clips per composer. The features used in their proposal [6] are based on processed timbral features from MFCC, visual features from the audio clip spectrogram, and temporal and visual features from statistical measures. They also perform a feature selection using the SVM ranker. Finally, the results are achieved with the classification with SVM (LIBSVM) with RBF kernel.

The same database was used in the MIREX edition 2012, where Lim et al. [7] achieved an accuracy percentage of 69,70%, one of he best results in MIREX competition in this task. His performance for this task involves timbral features, such as MFCC, DFB and OSC, and spectro-temporal features, combined with feature selection techniques. The classifier used in this study is SVM with RBF Kernel parameters.

2.3 Music Symbolic approach

Classical music, in contrast to other musical genres, usually is written in scores by the composers in order to be executed by the performers. In terms of music technology, there are several procedures to digitize the score information. One

[1] Music Information Retrieval EXchanges: annual competition of hot topics related to Music Information Retrieval (Audio beat tracking, audio tempo estimation, audio tag classification, etc.) where all the submissions are performed using the same blind database. The winner of each topic is the one that achieve better results in the respective tasks. http://www.music-ir.org/mirex/wiki/MIREX_HOME

of the most extended formats is the MIDI standard (*Musical Instrument Digital Interface*) [2].

The MIDI standard involves the exchange of musical information through files with **.mid* extension that have some related features for each note. These MIDI files are suitable to be played on a virtual synthesizer, which normally has a bank of 128 different sounds of the GENERAL MIDI specifications and 16 available channels (with channel 10 reserved for the percussion).

Although this standard is able to store and transmit information with a lot of options, we will focus on the most common ones, which we will need in our research case. Each MIDI file is organized by tracks, where each track belongs normally to one instrument or voice. Each track needs a channel to be played.

The MIDI files can be created by the transcription of a music score using several existing designing scoring software, such as *Finale* [3], or *Sibelius*[4], among others. Another way to create a midi file is using a MIDI keyboard connected to a DAW (Digital Audio Workstation) which records a musical performance as MIDI information in the exact place where the performer executes all the notes. The problem with this previous one is that the information is not human-interpretable, as it could be a piece from a score transcription.

There exist other formats in order to codify music as virtual information. Some reports [8, 9] use the *Kern* format [5], which is another representation of music scores which also contains information about dynamics, articulations and other musical features that are relevant for some kind of studies or music. On the other hand, there are several software applications based on Optical Music Recognition (OMR), like *Audiveris* [6] and *OpenOMR* [7], which are able to scan scores from images to Midi format or musicXML, which is another standard of digital sheet music.

2.4 Composer identification by pattern recognition

Some classical composers have their own chord progressions that are repeated over their whole work. This could be a clue for identifying them. The problem is that in some cases, these chord progressions are not only used by the composer, because they belong to one musical period, e.g. some typical *cadenzas* from the classicism.

[2] http://www.midi.org/
[3] http://www.finalemusic.com/
[4] http://www.sibelius.com/
[5] http://kernscores.stanford.edu/
[6] http://audiveris.kenai.com/
[7] http://sourceforge.net/projects/openomr/

Backer [10] proposes a feature set called *style markers* which are related in most of the cases to the musical periods of the history. He uses them in order to classify 300 different composition of 5 composers: Bach, Handel, Telemann, Mozart and Haydn. These features rely in the simplest analysis of music, as could be the interval distribution between voices or the pitch entropy, among others. The results that he has achieved with these experiments using a two-class classification (is a composer/is not that composer) are error averages around 4% and 9%, whereas the five-class classification, he gets confusions between composers very close in time (Mozart vs. Haydn) with error averages around 24%. However, the use of these features could be useful in order to see how the system works with our data set, which contains a larger range of composers.

2.5 Composer identification by melody

The melody of the pieces is one the most important part of the music for the majority of the people: this music component remains in our memory and we use it to recognize the work. However, due to the complexity of classical music, not always a work can be identified using the melody or either a chord progression.

Peusner [11] presents a graph topological representation of score melodies based on the repetitions of little segments called phonemes, which are the smallest combination of notes that can be constructed in a melody fragment. However, this graph only could be helpful in order to explore manually the melodies from the works, so there could be useful for the development of new descriptors based on the melody. These graphs have to be done manually, so there is not any software to perform them automatically. They are based on the repetition of note sequences taking or not taking into account the duration of these notes. There is an example in Figure 2.1.

On the other hand, Li et al. [12] uses the Kolmogorov complexity of 771 melodies of Beethoven, Haydn, Chinese music and Jazz music, distributed non uniformly, in order to classify them among these classes. Kolmogorov complexity [13] relies on which is the most short simplification that can be possible of a data set. As this complexity is not calculable, they use a similarity metric based on compressibility of data with common compressors as LZ78 parsing algorithm. They only use the pitch contour of the melody (with relative and absolute pitch as well) and then, they compute the difference between the extended melody and the compressed one. Once they get that value, they are able to perform the classification using three different classifiers: k-NN (k-Nearest Neighbor), SVM (Support Vector Machines) and SLM (Statistical Language Model). The best result achieved is around 92,4% using 1-NN classifier and Relative pitch. The highest accuracy in this study is because of the wide differences between the style

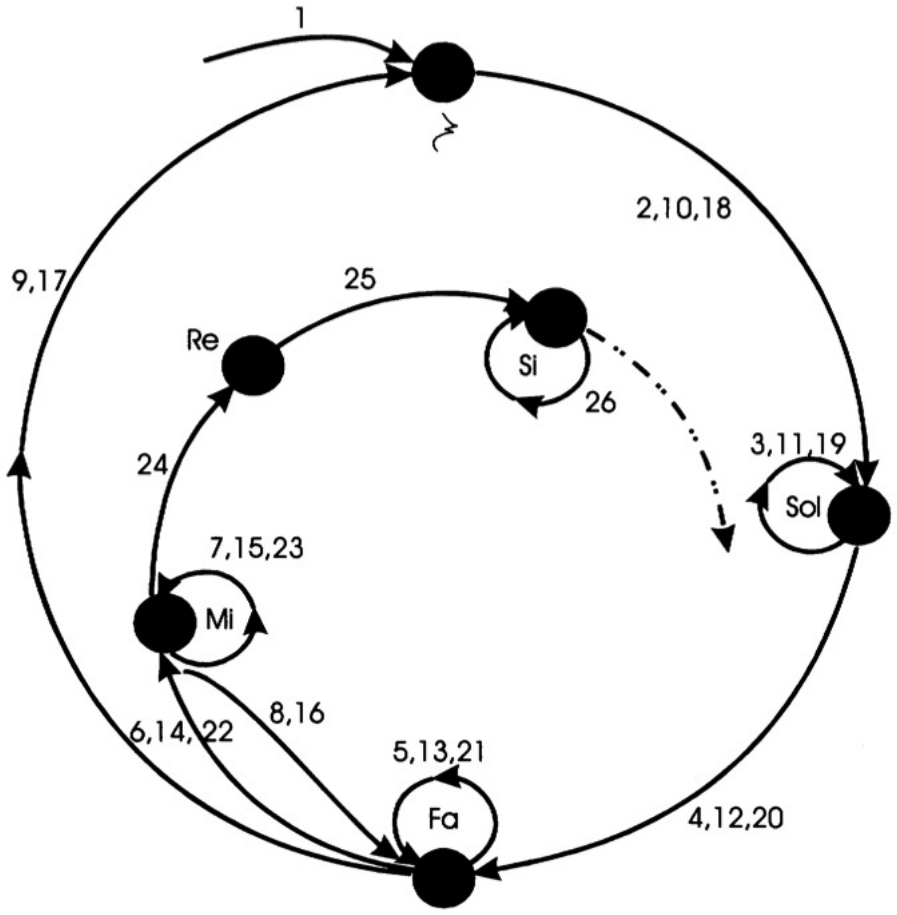

Figure 2.1: Graph topological representation: First few measures of Concertod Autunnoby C. Bargoni (1956). Numbers indicate the order in which transitions take place. Black dots represent the notes. (©KarinaZerillo-Cazzaro)

melodies analysed, which suggests they face a relatively easy classification problem.

Another report related to the melody analysis [14] uses Markov Models as the core of their classification models. In their test, the authors use 605 melodies equally distributed by Mozart, Beethoven, Dvorak, Stravinsky and Beatles, and the tests are performed using machines and humans as well. For the classification using computers, the information used is the interval distribution and the note durations from the melodies of the compositions. They created 28 different alphabets from the combining representations of intervals and durations. Then, they used a model from the research field known as *computational biology* [8] for three different

[8]Computational biology involves the development and application of data-analytical and theo-

states (match, insert and delete) in order to classify the instance among the different classes. The results from the machines tests, relies in the range 60% - 70%, whereas for a group of human non musical experts are around 24% whereas 48% of accuracy achieve by musical experts. This means that the computers improves the classification better than the humans, even in the case of musical experts; thus, the melody has an important information which defines the composer.

Hillewaere [15] converts string quartets by Haydn and Mozart into four separated melodies (one per each instrument) and constructing a newer one from the concatenation of the previous ones. For the classification he uses global feature sets from different studies (Alicante [16], Jesser [17] and McKay sets [18]) and n-gram models. The best results are achieved using 3-gram models and using Violin I and Cello melodies, achieving a accuracy classification of 79,4%.

Another study related to the composer classification using the melody, is written by Kaliakatsos-Papakostas [19]. In this one, similar to the previous one, the author classifies string quartets by Haydn, Mozart and Beethoven, but classifying only between two classes at a time, performing the three possible combinations. He treats the string voices like in the previous paper (but without the concatenated voice), and then he applies a simple Markov chain model for the classification. Then, he compares the results with the classification achieved using weighed Markov chain model, where the best improvement is in the classification of quartets by Beethoven and Haydn, using the 1st Violin voice resulting a 66% of accuracy percentage in the former one versus a 88% in the latter one.

2.6 Composer identification in other genres

As we already said, the composer identification is normally performed to classical music. However, there are some other reports that relate to the composer identification of musics that don't have the complete score with the information of all the instruments and players. This is the case of the Jazz music, where we normally find in the scores the melody of the pieces with several annotation related to the accompaniment and the harmony that have to be played with the melody and some clues about the rhythm. Therefore, this musical information cannot be treated the same way than classical music in order to identify the composer of a piece.

T. Hedges et al. [20] propose several experiments for classifying jazz composers and discuss the results achieved by their research study. The dataset used by them is from the *Real Books* of Jazz standards, where they create 4 subsets in order to perform different classification tasks by: composer, sub-genre, perfor-

retical methods, mathematical modeling and computational simulation techniques to the study of biological, behavioral, and social systems. ["NIH working definition of bioinformatics and computational biology". Biomedical Information Science and Technology Initiative. 17 July 2000]

mance style and meter. They create an harmonic dictionary in order to set for each chord their root among the 12 possible pitches and the most closer simple chord among dom, maj, min, dim, aug, hdim and NoChord (when the chord doesn't fit in any one of the other classes). Related to the melody, from each note of note sequence, they extract its absolute pitch and its duration. Then, they perform several classification experiments: supervised classification of chord sequences (separate classification of chord progressions), supervised classification with multiple viewpoint classifiers (4 different combinations of parts of the extracted information) and classifying subsequences within compositions (segment the pieces into several parts, and classify all them as individual excerpts). The most prominent results achieved are in the combination of multiple viewpoint, where they get an average accuracy around 67% of the cases. However, this accuracy is different depending of the classification experiments.

Related to the previous study, the harmonic simplification could be useful in our study because it could give us information about which are the most common chord patterns of each composer. However, in some many cases, mostly in the most modern periods of music, the chords are very complicated and we could be not able to identify them as the simple version of these chords. This could lead to having an important number of *NoChord* class that could give us biased results or wrong results. We will use the concept of NoChord in our study for the cases where the combination of notes won't fit with the classification or the chord profile that we are using in our feature extraction.

2.7 Other composer identification approaches

The following reports present different procedures used to achieve the same goal: identify the composer of a work using the symbolic information allowed in any virtual file. The differences between them are that they create a different subset of composers, with a very specific target.

In a study by Mearns et al. [9], the authors use a database with works from the baroque period. In these works the most used technique is the counterpoint, i.e. the musical relationship between the several voices that integrates the whole polyphony or chord progression of the baroque music. The descriptors are created in order to extract information related to intervals between the voices (perfect consonance, imperfect consonance, dissonance) and the motion between them (parallel, similar, contrary oblique). The values from these descriptors are classified using Naive Bayes and J48 decision tree, achieving results of 66% of well classified instances among a data set of 66 pieces and 7 composers. These composers are from the same period, so their music has similar composition techniques and style, which makes the problem more difficult.

On the other hand, other studies classify music between only two composers. In Y. Liu report [21] he explains how to create a Markov model based on the differentiation between the two composers. The core of his method relies on the creation of space distances between two different models, following several steps and computations. Then, with the model already built, he is up to run the experiment between a data set consisting of works from the two composers for what the models were created. In the case of Mozart vs. Haydn string quartets, he achieves results about 65% of accuracy. Although this accuracy doesn't seem to be relevant because of it is binary classification, both composers belong to the same musical period, so their music has similar elements what could imply that the classification has some difficulty.

There is a report based on the classification of notes, ignoring everything except the pitches and their durations [8]. As they are *kern* files, each of their lines corresponds to a chord which will be classified with a previous transposing of the piece to C major, in order to be able to compare them. The works belong to 7 composers, one of them with two classes (keyboard and string quartet), with a different number of works for each composer. The transposed chord progressions are run in 4 different classifiers with their particularities: Naive Bayes, SVM, Linear and Quadratic Discriminant, and K-NN. As a conclusion and results of their experiment, the best classifier is SVM with an accuracy error between 10% and 60%, whereas the other ones even have an error of 100% for a few cases.

The Dodecaphonic Trace Vector [22] has the particularity that is useful independently of which is the formation of the piece or the tonality, because it normalises the score to a set of Chromas and then normalizes all the notes to the most present one. The notes are placed in a 12-dimensional vector (on for each chroma class) which contains the durations of these notes, normalized depending on the duration of the work. Then, the piece is also transposed to C major or A minor (his minor relative) in order to be able to compare all the works and extract some characteristics of each composer that made him unique. Finally, the author uses Neural Networks to identify the composer.

2.8 Summary

We have exposed several ways to classify music among the composer. In most of the cases, these studies focus on the classification of a reduced number of composers: 8 or 10 at maximum. If we take a look to the results achieved by these studies, we see that the larger is the number of composers to classify, the lower are the percentages of good classifications. In addition, if the group of composers belongs to the same musical period, that task is even more difficult and the results tend to be worse than composers from different moments. One of the innovative

aspects of this study that contrasts with previous ones is that we will perform the classification of a dataset around 10.000 works by more than 100 composers.

From the studies that we have commented, we can use in our experiments, some characteristics that could be interesting in order to create new features (like observe the melody of the works or transpose the works to C Major or A Minor), but we plan to create new features in order to achieve our goal.

If we compare between audio and symbolic approaches, we think that we may take more profit of the symbolic approach. In this approach, we will be able to extract from the notes that composers wrote, so we could use musical theory in order to create proper descriptors to find characteristics that defines composers essence and single writing styles. On the other hand, we know that in classical music the performances of the players sometimes are quite different between them, so we will like to avoid these differences; using the score information the personal performing style of the musicians is ignored.

Chapter 3

METHODOLOGY

In this chapter we will expose all the methodology of this study, including the experimental set-up, the database and its filtering, the features design and the classifying configuration.

The experimental set-up of this thesis involves a classic feature extraction and final classification, used in several of MIR tasks. In Figure 3.1 there is the basic scheme with different steps needed for classification tasks of this study. Following sections includes an extensive presentation of the database used in this study as well as the detailed explanation of each step that will be used in the classification scheme.

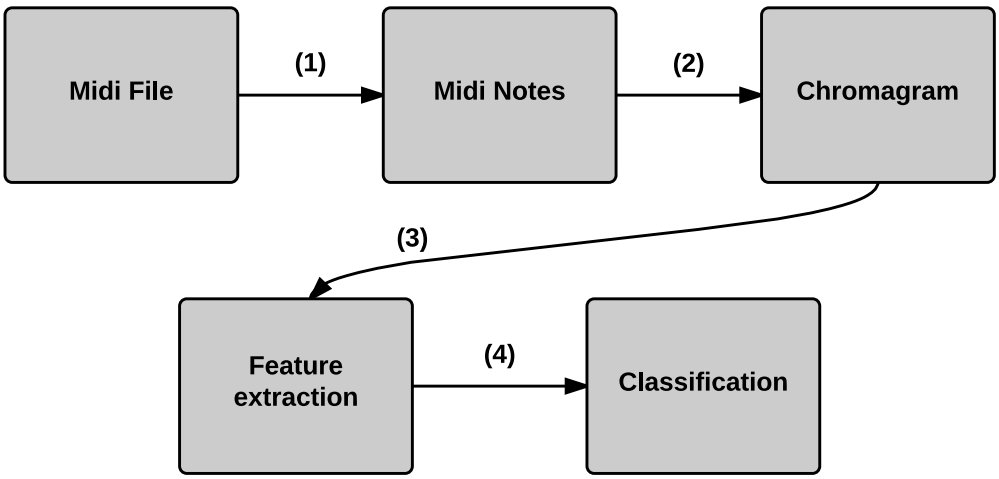

Figure 3.1: Process scheme that will be used for the classification tasks. 1: smf2txt converter, 2 & 3: python scripts, 4: sklearn package

The master programming language that we use in order to manage the information from the database is Python 2.7 [1]. However, in some of these steps we use external applications or packages to convert or classify the information.

For the explanation of the different parts we will follow the chronological order that we have followed in this study.

3.1 Database

The database that we use in this study is *Kunst der fugue* [2]. This database is a collection of MIDI files by several composers, with any kind of musical ensembles (piano solo, string quartet, orchestra, etc.). This data set is a mixture of transcribed MIDI (directly from the scores) and DAW MIDI (recorded performance with a Digital Audio Workstation from a MIDI controller, typically a piano keyboard).

The original database contains 16961 midi files by 894 composers. It includes musical works since the Renaissance (15th century) until the Twentieth Century.

We need to filter the database taking into account several parameters. The first filtering that we perform is hand-operated. In this step we use basic folder browsing, i.e. take a look to names of the files and folders, and listen some of these files in order to compare them.

First of all, we remove all the folders that contains the word *live*, which mean the these files are recorded by a virtual player. We remove these files because they are susceptible to have wrong notes. We also remove a folder called *piano rolls*, because it contains several transcriptions of orchestral works that we don't want to use in our study.

The second step is looking for different versions of the same work and remove one of them. The procedure to find duplicated works is performed browsing the folders that contain catalogued works from the composers. In these folders, we apply basic searching tools using the catalogue numbers of the works as key words; E.g. BWV catalogue for Bach works. In cases of repeated works we keep the one that has the movements of the work split by several files because in some cases, these movements have different music keys between them, which will be used for the feature extractions in following sections.

In all classification tasks, a minimum threshold of training samples are requested in order to perform a classification. In this study we set-up a minimum of 20 works per composer presents in the database in order to that composer be part of this study. We perform this classification automatically over the manual filtered database, discarding all the works from composers that have less than 20 works in the database.

[1] https://www.python.org/

[2] http://www.kunstderfuge.com/

Now we are able to load files from the filtered database in order to perform a second filtering. However, we are not able to read directly from the midi files, so first we will extract the notes information using the procedure explained in the following section.

3.1.1 Midi converter: smf2txt

In this section we will explain the step 1 of the Figure 3.1 that involves the conversion from midi files to midi notes.

Once we have filtered the database, we are able to load files in order to be processed by our algorithms. Midi files store information of notes with a codification that we are not able to process directly from the files. We use *smf2txt* [3][23] converter.

This converter extracts from a midi file the requested information related to its notes, and writes them to a text file. This new file contains a header of the file characteristics and the ticks resolution (quantization) of the file. Notes information are grouped track by track, with a sub-header that includes the name of the track followed by the note's information in each single line of the text.

Each file has a ticks resolution number, linked to the temporal quantization of the notes. This number sets up the relative duration of one beat in the file. The tick is the temporal unit for note onsets and durations. E.g.: A note with an onset of 2048 and a duration of 512, with a resolution of 1024 means that this note starts at the beginning of the 3rd beat and it longs a half of one beat. The resolution number will be very important in the next sections: the beat will be the basic temporal unit of our files, and the resolution is the number of ticks per beat.

The application has many extraction options. The option that we use in this study extracts from each note of the midi file the following information:

- Midi pitch (number from 0 to 127): Related to the pitch of the note, where the A440 Hz corresponds to 69. (Complete numbers table in Figure 3.2)
- Onset (ticks number): Related to the *quantized* emplacement of the starting point of the note.
- Duration (ticks number): Related to the *quantized* number of ticks that extends the note.
- Velocity (number from 0 to 127): Related to the dynamics of the note. The higher the number, the louder the note sound, and vice versa)
- Channel (number from 1 to 16): Related to midi channel that plays the note.

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Figure 3.2: Midi pitch numbers, where the number 69 corresponds to A440 Hz

In Figure 3.3 there is an example of the conversion of 3 single notes to text using a resolution of 1024 ticks per beat.

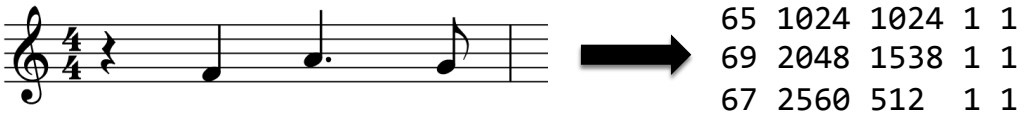

Figure 3.3: 1st step conversion example: 3 notes conversion to text, with resolution of 1024 ticks per beat

Once we are able to process all notes from the midi file, we perform a second filtering, based on the number of notes and tracks of the file. In this case, the filtering has been applied after the conversion from midi files to midi notes (step 1). We dismissed the files using three different filters:

- Midi files that can not be loaded by the converter because of their original potential wrong encoding.
- Midi files with more than 50 tracks. These files may be wrong encoded because we don't find works with that high number of different voices in classical music.
- Midi files with less than 40 notes.

After all these filters, we have the database properly filtered and we can start using it in the experiments.

[3]http://grfia.dlsi.ua.es/gen.php?id=resources

3.1.2 Final Database

With the application of all the exposed filters, the database has been significantly reduced.

In this study, we perform our experiments based on 3 composer groups setting different threshold works to the groups. These ones are formed by composers with more than 20 works (complete filtered database), 50 works and 100 works in the filtered database.

We also classified the composers depending on 5 general historical periods: Renaissance, Baroque, Classic, Romantic and 20th Century.

In Table 3.1.2 there are the distributions of works by composers and periods of the mentioned classification groups.

Works threshold	20 Works	50 Works	100 Works
Number of composers	106	47	17
Number of works	9876	8069	5911
Maximum num. works	1389	1368	1368
Minimum num. works	20	50	109
Num. works mean (std.)	93.2 (165.6)	171.7 (225.1)	347.7 (302.0)
Num. Renaissance comp.	15	10	2
Num. Baroque comp.	47	17	6
Num. Classical comp.	9	3	3
Num. Romantic comp.	25	14	6
Num. 20th Century comp.	10	3	0

Table 3.1: Filtered data base characteristics with the 3 threshold works groups used in this study.

In Appendix A there is a complete list with all the composers that are part of the database with their number of works.

Once we have the database properly filtered, the next step is to transform notes information from the midi files of the datbase, into tables called *Chromagrams* where we condensed all the information of the each score in one single table. We will present it in the following section.

3.2 Chromagrams

This section is related to the step 2 of the Figure 3.1 that is the conversion from the midi notes into a Chromagram.

In order to group all notes information condensed into a single table, we use the Chromagram. This is technique where we will condense all the information of

the score into a single octave (12 chroma classes) and with an specific reference duration. We think that the information related to the composers identification is present in the chroma class distributions, chord progressions and sequences that he tends to use. The Chromagram is an extensively used mechanism to capture and analyse these characteristics.

Each midi file has a single Chromagram, which is a table where the X-axis is related to the temporal distribution of the notes, whereas the Y-axis is related to the pitch distribution of them. About the Y-axis, each row represents a *chroma class* (C, C#, D, D#, ..., A#, B). Using the relations of the Figure 3.2 and the midi pitch of each note, we assigned a chroma class to each note, i.e. we assigned in which row we have to locate the note value. The number of rows per Chromagram is always 12 (one per chroma class). Related to the X-axis, each column represents one beat of the midi file, i.e. each column longs the number of ticks (resolution number) that we have obtained from the midi conversion in the previous step. The placement of the note value depends on the note onset, which tell us the starting point of the note. Then, the note duration is represented by the value, where a value of 1 corresponds to the duration of one beat. The maximum value per cell is 1. In the case that a note longs more than 1 beat, the remaining duration have to be located in the following column. The number of columns per Chromagram is the number of beats of its related midi file.

Each column of the Chromagram has to be normalized to 1 by the higher value of the column (excepting where there is not any note during all that beat where all the values remains to 0). Before the normalization, some cells could have values higher than 1, which means that there are several notes of the same chroma class that belongs to different instruments or octaves. On the other hand, some cells could have values lower than 1, which means that in that beat, the note longs less than the whole beat.

In Figure 3.4 there is a detailed example, step by step, of the Chromagram construction. This example is the continuation of the example of the Figure 3.3, where we transform 3 notes into a Chromagram that longs 1 bar (4 beats).

One aspect that we need to take into account is that we have to dismiss the notes played in the channel 10. This one is always reserved for unpitched percussion instruments only, so the chroma class information allowed in this channel doesn't have any relationship with the chroma classes of the other channels. We simply don't incorporate the notes played in this channel to the Chromagram.

The Figure 3.5 is an example of the complete Chromagram of a work by J.S. Bach. The visual inspection of these tables doesn't give us so much information about the work, but for instance, in this explicit case we could suppose that the main key of this work is G Major because most of the notes that are present in the Chromagram belong to this scale.

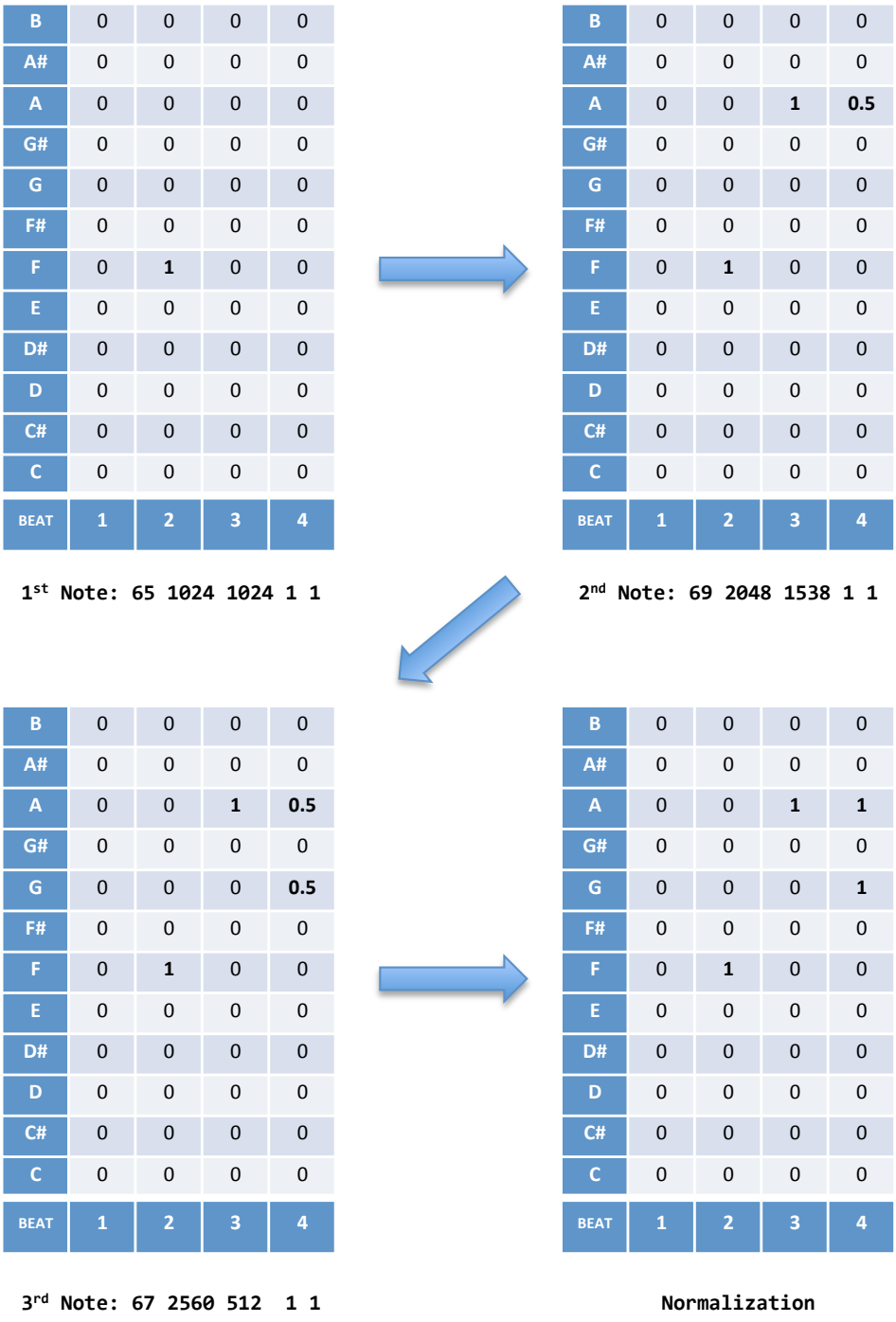

Figure 3.4: 2nd conversion step example: 3 notes incorporation to Chromagram, with column normalization, which only impacts on the 4th beat.

Figure 3.5: Chromagram example: Minuet in G Major BWV Anh 114 - J.S.Bach. The more darker the cell, the more present the note. Vertical axis: chroma class; Horizontal axis: beat number

3.3 Features

Features are the information that we extract from the music scores and give us an specific characteristic of the compositions. These features are based in several aspects that we will explain in the following subsections.

The features extracted from the scores are designed for this study and are presented in the form of numbers or vectors. The features are extracted directly from the Chromagrams created in the previous section. Once we have computed these extraction, we store the values from each feature in separate folders per each file. Then, we will use these feature values in order to perform a composer classification.

Before presenting all the feature groups, we will explain how we performed the key estimation and the structure of the chordgrams, both used in some feature extractors as well.

3.3.1 Key estimation

Some of the features extractors that we designed in this study need the main key of the work in order to extract properly the information.

For the key estimation we use 2 key profiles discussed in a study by Temperley [24]. Each profile includes two vectors of 12 values per vector, which represents the chroma classes distribution of the major mode and the minor mode of a work, respectively. One profile is developed by Krumhansl, whereas the remaining one is proposed by the author himself.

In order to estimate the key from a work, we compute the mean of each chroma class of its Chromagram, resulting in a 12 values vector. Then, we calculate the Pearson Correlation over the 12 shifted positions, which corresponds to all the chroma classes, of the proposed key profiles (major and minor modes). The shift of the highest correlation number and the mode of the profile which achieved it tell us which is the estimated key. We perform the key estimation separately for each author profile (Temperley and Krumhansl) in order to compare the resulting keys.

In the analysis of both key estimations, we have found the following groups based on the relations between them:

- **Coincident estimated key (5835 works):** Both profiles presents the same estimation.

- **Relative confusion (1683)** When one profiles estimates a key, the other profile estimates its relative one. E g. Temperley = G Major, Krumhansl = E Minor.

- **Tonic/Dominant confusion (1504 works):** When Temperley profile estimates the tonic of the scale, Krumhansl one estimates the dominant of it. E.g. Temperley = G Major, Krumhansl = D Major.
- **Third confusion (499 works):** When Temperley profiles estimates a major mode, Krumhansl one estimates a minor mode from the third major higher from the Temperley estimated root. E.g. Temperley = F Major, Krumhansl = A Minor.
- **Mode confusion (178 works):** Both profiles presents the same root but with different mode. E.g. Temperley = A Major, Krumhansl = A Minor.
- **Others (177):** When Temperley and Krumhansl profiles estimates different keys and no fits in the previous groups.

In order to choose the correct profile, we checked the estimated key manually from all the previous groups. In the case of coincident estimated keys, as we cannot manually assess the reliability of the estimated keys, we use 40 random works that are examined manually and infer, from that, the overall reliability. From these cases, we found 1 mode estimation error, 1 root estimation error and 2 key estimation errors (mode and root were incorrect). The remaining 36 works were estimated correctly. Related to the other groups, we check 20 random works per group and we obtain the results shown in Table 3.3.1. Taking into account the checking, we decide to use the Temperley estimation in all the cases, except for the group of relative confusion, where we take the Krumhansl estimation.

Correct pf. / Conf. gr.	Relative	Tonic/Dom.	Third	Mode	Others
Temperley	1	19	7	13	9
Krumhansl	17	0	3	6	5
None	2	1	10	1	6

Table 3.2: Key estimation checking. The values are the number of works with correct estimated key by each profile. If any profile achieves a proper estimation, then the number work is located in the *None* category.

We add the estimated key to the file that contains information of each work of the database, including the mode and the root of the estimation.

3.3.2 Chordgram

In a musical work, there are notes that are played simultaneously, producing different combinations of sounds. These combinations can be grouped in unisons

(combination of the same notes), intervals (two different combined notes) or chords (three or more different combined notes) [4]. However, these combinations are only relative, i.e. they only take into account the distance between the notes, not the absolute pitch were are placed.

One chord can be expressed with the combination of two elements:

1. **Root:** Chroma class name that indicates where the main note of the chord is placed. E.g. "***D** Major*"

2. **Name:** Nomenclature that give us the relationship between the component notes of the chord. E.g. "*G **Minor7***"

The root tends to be located in the lowest part of the chord, but it needs not be the bass note of the chord: the concept of root is linked to that of the inversion of chords; i.e. the lowest note of the chord can be another component of the chord. Using the information from the Chromagram, we can not obtain the root of each chord. Because of this, in some cases, there is a one single binary chord vector related to two different chords. This fact is due to the roll shifting of the binary chord vector that we perform when we look for vector that fits with the chord. E.g. A chord made by the notes C, E, G, A could be C M6 or A m7. In this study, we use a chord dictionary that contains most of the chords used in music; however, we group the chords with the same chord shape taking into account the approximation to the root mentioned in the previous lines.

On the other hand, we have in consideration all the intervals combinations between two notes (2nd minor, 2nd major, ..., 7th Major) and perfect unison as well. In order to use them, we transform all the chords into a binary vector of 12 value notes (12 chroma classes), where a 1 means that the chroma class is in the chord, whereas a 0 means that is not present. In Appendix B there is information related to the chord dictionary B.1 and the table of binarized chord vector profiles B.1, which includes the chord profile vector, the chord name and the simplified chord name per each chord.

A Chordgram is a mapping that we perform from a Chromagram. Each column of the Chromagram corresponds to the distribution of all chroma classes during a beat of the work. The procedure to transform a Chromagram into a Chordgram is:

1. Binarize the Chromagram using a threshold; i.e. the values of the Chromagram that are higher or equal to the threshold became 1, the lower values became 0. Now, the Chromagram is a table of 1 and 0, where each column is a binarized vector of 12 values, corresponding to one beat.

[4] In this study we will consider unisons and different intervals as a type of chords, just in terms of nomenclature.

2. For each column we have to find the chord profile that fits with each column. We have to compare the vector from the column with the 12 roll shifted positions of the chord profiles of the chord dictionary. Each shifted chord profile is unique in all the dictionary. In Figure 3.6 there is an example of 1 and 2 position shifts of the original chord profile.

3. When we find the chord type and shift that fits to the column that we are checking, we keep the information of the chord type and the shifted number. If we don't find the proper profile, we set to that column NC category (No Chord).

Figure 3.6: Roll shift example of the C Major profile. In the first shifting, the chord is C# Major and in the second one is D Major.

In this study we create two Chordgrams using different thresholds: one with the threshold in 0.5 and the other one with the threshold in 0 (this case means that any value higher than 0 becames 1). The former creates the Chordgram using the most remarkable notes of each beat, in order to avoid possible secondary notes or passing notes; the latter creates the Chordgram using all the notes of each beat whatever it is their importance during the beat duration.

As a summary of the Chordgram, we transformed the information of the Chromagram, where each column is a vector of 12 values from the distribution of the chroma classes per beat, into a Chordgram, where each column has two fields: chord type and shifted position number (from 0 to 11).

3.3.3 Features design

In this part we will explain which features we developed for this study. These ones are based in musical and statistical aspects of the scores, trying to extract different characteristics from the music compositions.

These features have been designed considering that all the works have different durations. Because of this, they are based on histograms or class statistics, are normalized with the beats number of the piece.

On the other hand, some features are related to the relationship or differences between beats, giving as a results a vector with length determined by the duration of the piece. In these cases, we computed six statistical measures to these vector, which we use in the classification. We computed the statistical measures that are usually used in this field:

- Mean
- Standard deviation
- Kurtosis
- Skewness
- 95th percentile
- 5th percentile

Features that are expressed using these statistical measures will be presented with the sign * following their title names in their respective presentations. E.g. Internal Correlation *

Chroma vector (binary_vec)

This feature is based on the mean distribution of the chroma classes during all the work duration.

For the computations of this feature, we use the estimated key and we transpose to the reference modes (C Major or A minor). The transposition is achieved rolling over the rows of the Chromagram depending of the semitones difference number between the root of the key and the reference mode. E.g. A work with an estimated key of E Major have to be roll shifted down 4 positions (until C Major), whereas a work with an estimated key of G minor have to be roll shifted up 2 positions (until A Minor). Then we perform a binarization of the Chromagram setting a threshold of 0.5 (the higher values became 1, the lower values became 0). Finally we compute the mean over each row, getting as a result 12 values, one per each chroma class.

Number of feature values: 12

Chromagram histogram (histogram_vec)

This feature give us information about the complexity of work beats, and their distribution during all the work long.

The procedure to extract this feature starts binarizing the Chromagram with a threshold of 0.5. Then, each column is a vector of 12 binary values. We merge each of these vectors in order to construct several binary numbers, which we convert to decimal system. E.g. [1 0 0 0 1 0 0 1 0 0 0 0] = 2192. Finally, we perform the normalized histogram of all the numbers that we calculated. As there are 12 values per each vector, the number of bins of the histogram is 2^{12} = 4096.

Number of feature values: 4096

Empty bins histogram (empty_hist_bin)

This feature is based on the previous one. It also give us information about the complexity of the work.

From the histogram calculated in the previous feature, this one computes the number of bins from the histogram that remains to 0, i.e. the number of empty bins in the histogram.

Number of feature values: 1

Profile template correlation number (profile_template_correlation)

This feature presents the similarity between the key profile used in the key estimation and the Chromagram mean pear each chroma class.

To extract this feature we transpose the Chromagram to the reference modes like in previous features. Then, we perform the mean of each chroma class over the Chromagram. Finally, we compute the Pearson correlation between the key profile used in the key estimation and the vector from the Chromagram means. We use the central number as the feature value.

Number of feature values: 1

Internal correlation *

This feature give us the similitude between the beats of the Chromagram, taking into account the chroma classes distribution over the whole beat; i.e. the more similar are the compared beats of the Chromagram, the higher is the feature value.

The extraction of this features consist on the correlation between columns (beats) of the Chromagram. In this feature we use different lags. E.g. In lag = 1, we compare column 1 vs. column 2, column 2 vs. column 3 and successively, where in lag = 2 we compare column 1 vs. column 3, column 2 vs. column 4 and so on. We perform this feature using 1, 2, 3 and 4 lags.

For the feature extraction, we perform the Pearson correlation between all the columns of the Chromagram following the lag instructions that we explained before. The correlation between two columns returns one single value. We compose a vector using the values from the correlation between all the columns comparing them pair to pair. Once we finished all the correlation for one single lag, we compute the statistical measurements of the correlation vector and we store these measurements. Now, repeat all the procedure for the next lag since we finish all the lags.

*Number of feature values: 6 * 4 lags = 24*

Differential *

This feature computes the amount of energy changes between beats of the same chroma classes. This give us information about the duration of the notes.

The extraction process begins applying the differential over the columns of the Chromagram. Now, we perform the absolute value of all the table and we compute the sum of all the rows (the resulting vector has the same length than beats of the Chromagram less 1 value). We perform the statistical measurements to this vector. Then, we repeat the same operation but without apply the sum of all the rows, i.e. we keep the differential of each chroma class separately. We compute the statistical measurements to each of the differentials of the chroma vectors and we add them to the feature information.

We perform this feature two times in separated groups: one with the original Chromagrams and another one applying binarization with threshold at 0.5.

*Number of feature values per feature group: 6 (together) + 12 (separated) * 6 = 78*

Sumatory columns *

This feature give us information about how much dense are the beats of the works. The more dense is a beat, the higher is the sum of that beat.

The feature extraction is based on the sum of each column of the Chromagram. Finally, we compute the statistical measurements to the resulting vector and we store these values.

Number of feature values: 6

Chroma class beat presence (beat_pres_chroma)

This feature performs a histogram related to density of each beat of the Chromagram; i.e. the more dense (number of present chroma classes) is a beat, the more complex is the respective chord of that beat.

The process of extraction consists of walk over the columns of the Chromagram and create a normalized histogram of the number of 0 presents in each column. The minimum of 0 in a column is 0, and the maximum is 12, so the histogram has 13 bins.

We perform this feature in two ways (two separated groups): one with the binarized Chromagram using the threshold of 0.5, and the only one only keeping the values that were 1 at the beginning (stronger values).

Number of feature values per feature group: 13

Combinations histogram (comb)

This feature performs a histogram of the combinations of each beat of the Chromagram using the chords profiles (Appendix B).

The procedure is to find the chord profile that fits with each column of the Chromagram and then perform a normalized histogram of the chord. Each bin of the histogram belongs to one chord profile independently of the shifting needed to fit it.

We perform this feature in two ways: one with the binarized Chromagram using the threshold of 0.5, and the only one only keeping the values that were 1 at the beginning (stronger values).

Number of feature values per feature group: 50

Intervals between strongest unisons (interval_unison_val)

This feature aims to locate the lead voice of the score, that should be on the strongest values of beats. Then, it looks for the horizontal interval (interval between two unisons played non simultaneously), i.e. the distance between single chroma classes of different beats. If any beat has more than one simultaneous note, then we miss that beat. In this feature we perform the differences with 4 lags, with the same procedure that we explained in the Internal Correlation feature.

The possible interval classes of these notes are:

- **Perfect unison:** The notes are the same.
- **2nd minor or 7th major:** Distance of semitone or five tones and semitone.
- **2nd major or 7th minor:** Distance of one tone or five tones.
- **3rd minor or 6th major:** Distance of one tone and semitone or four tones and semitone.
- **3rd major or 6th minor:** Distance of two tones or four tones.

- **Perfect 4th or perfect 5th:** Distance of two tones and semitone or three tones and semitone.

- **Tritone:** Distance of three tones.

The extraction of the feature begins with keeping only the stronger values from the Chromagram, i.e. the values that are already 1. Then, we calculate the distance between chroma classes from the beats that we extract the interval (depending of the lag). Finally, we normalize the number of elements per class depending of the work duration.

*Number of feature values: 7 * 4 lags = 28*

Interval evolution (rel_interval_val)

This feature compares the vertical interval (interval between two notes played simultaneously) between two beats with less than 3 chroma classes. If any beat has more than two simultaneous notes, then this beat is missed into account. In this feature these differences with 4 lags using the same lag procedure that in previous features.

The categories of intervals used in this feature are the same than in the previous one. However, we compare these intervals between 2 beats, so create $7^2 = 49$ per lag. E.g. Perfect unison to Tritone, 3rd minor or 6th major to Perfect 4th or perfect 5th, etc.

This feature is similar to the previous one. We have to keep only the stronger values from the Chromagram as well. Then, we calculate the vertical interval from each of the beat that we have to analyse (depending of the lag). We increase an unit on the proper evolution group. Finally, we normalize the groups depending of the work duration.

*Number of feature values: 49 * 4 lags = 196*

Chords relation (relation_chord)

This feature extracts from each beat the chord type and distributes them in small groups depending of the chosen lag and the number of chord classes.

The lag in this feature is not used in the same way than in previous features. In this feature, when we set lag = 1, we distribute chord types for consecutive pairs of beats; when we set lag = 2, we distribute chord types for groups of three consecutive beats, but including in the feature group the distribution using lag = 1; i.e. each lag for the feature group includes all their predecessors with minimum of lag = 1.

Related to the chord classes, we use 2 different chord classification in this feature:

- **6 classes:** Major (M), Minor (m), Diminished (dim), Augmented (aug), Suspensive (sus), No Chord (NC). In this case, all chord types with less than 3 notes belongs to NC class.
- **13 classes:** The 6 classes from the previous item plus the 7 interval classes exposed in Section 3.3.3.

In the extraction process of this feature we use Chordgrams. We have to map each chord type, which belongs to one the second column from the chord dictionary (Table B.1) into its correspondent simplification (third column from the same table). Depending of the number of classes used, unisons and intervals are NC, instead of the simplification. Then, we have to compute the distribution taking into account the lag used in the feature group and creating the chord groups like we explained before. Finally, we have to normalize the distribution according to the work duration.

In Table 3.3.3 there are the specifications from all the groups derived from this feature structure, as well as the number feature values, which depends of these parameters.

Chromagram threshold	Num. classes	Lag	Num. features
0.5	6	1	$6^2 = 36$
0.5	6	2	$6^2 + 6^3 = 252$
0.5	6	3	$6^2 + 6^3 + 6^4 = 1548$
0.5	6	4	$6^2 + 6^3 + 6^4 + 6^5 = 9324$
0.5	13	1	$13^2 = 169$
0.5	13	2	$13^2 + 13^3 = 2366$
0.5	13	3	$13^2 + 13^3 + 13^4 = 30927$
0	6	1	$6^2 = 36$
0	6	2	$6^2 + 6^3 = 252$
0	6	3	$6^2 + 6^3 + 6^4 = 1548$
0	6	4	$6^2 + 6^3 + 6^4 + 6^5 = 9324$
0	13	1	$13^2 = 169$
0	13	2	$13^2 + 13^3 = 2366$
0	13	3	$13^2 + 13^3 + 13^4 = 30927$

Table 3.3: Groups derived from chord relation feature structure

Chord simplification (chord_simp)

This feature simplifies the chord types from Chordgrams using the simplifications between the second and the third column of the Table B.1. Then, it computes an histogram of the chord classes of the simplified Chordgram.

In Table 3.3.3 there are the specifications from all the groups derived from this feature structure, as well as the number feature values, which depends of these parameters.

Chromagram threshold	Num. classes	Num. features
0.5	6	6
0.5	13	13
0	6	6
0	13	13

Table 3.4: Groups derived from chord simplification feature structure

Chord shift numbers (chord_shif_num)

This feature es related to the shift that was required to fit the correct chord profile when we created the Chordgrams using the Chromagrams. This number could be an approximation to the root of the chord, but as we already mentioned, it doesn't happens for all the cases because different chords with different root may have exactly the same chord profiles. However, this feature give us information about musical distances between chords.

Like in previous features, we create group features using this feature description with different specifications. In this case these specifications are the used lag, the threshold used in the Chromagram for the Chordgram creation and the transposition.

The lag concept used in this feature is the same than the one used in the previous features, but in this feature we find some cases with lag = 0: this means that we are only taking into account one chord by itself, without any neighbour relationship.

The threshold specification is the same than in the previous feature: Chordgrams from Chromagrams with threshold set to 0.5 and 0.

The transposition specification means that when we extract the feature from the Chromagram, first we transpose the key of the work to the reference modes using the estimated key. The categories used in for the distributions analysis corresponds to the 12 numbers of the shifts and another number that corresponds to the NC chord.

In Table 3.3.3 there are the specifications from all the groups derived from this feature structure, as well as the number feature values, which depends of these parameters.

Chromagram threshold	Transposition	Lag	Num. features
0.5	No	0	13
0.5	No	1	$13 + 13^2 = 182$
0.5	No	2	$13 + 13^2 + 13^3 = 2379$
0.5	Yes	0	13
0.5	Yes	1	$13 + 13^2 = 182$
0.5	Yes	2	$13 + 13^2 + 13^3 = 2379$
0	No	0	13
0	No	1	$13 + 13^2 = 182$
0	No	2	$13 + 13^2 + 13^3 = 2379$
0	Yes	0	13
0	Yes	1	$13 + 13^2 = 182$
0	Yes	2	$13 + 13^2 + 13^3 = 2379$

Table 3.5: Groups derived from chord shift numbers feature structure

Transposed Chordgram histogram (hist_transposed_chord)

This feature uses all the information from the Chordgrams: chord type and shift number. It performs the computations of the distribution of chords setting a specific lag, following the specifications of lag in Section 3.3.3.

The distribution groups are similar that the two used in previous features but with some variances:

- **6 classes:** Includes all the combinations of 12 possible shifts with all the previous classes (M, m ,aud, dis, sus) excepting of the class NC that has no meaningful pitch, i.e. 12 shifts * 5 classes + 1 NC class = 61 classes.
- **13 classes:** Includes all the combinations of 12 possible shifts with all the previous classes (unison, intervals, M, m ,aud, dis, sus) excepting of the class NC that has no meaningful pitch, i.e. 12 shifts * 12 classes + 1 NC class = 145 classes.

In Table 3.3.3 there are the specifications from all the groups derived from this feature structure, as well as the number feature values, which depends of these parameters.

Summary

All these features have been created as an intent to extract different characteristics from the music scores. That means that, even some of them may not have much sense if we only use that single feature group, if we merge the information from

Chromagram threshold	Num. classes	Lag	Num. features
0.5	6	0	61
0.5	6	1	$61 + 61^2 = 3782$
0.5	13	0	145
0.5	13	1	$145 + 145^2 = 21170$
0	6	0	61
0	6	1	$61 + 61^2 = 3782$
0	13	0	145
0	13	1	$145 + 145^2 = 21170$

Table 3.6: Groups derived from transposed Chordgram histogram feature structure

some features of different groups, we would improve the results achieved in our task.

In the following section we will explain how we performed the classification of composers using the information from these features.

3.4 Machine learning classifiers

This section is related to the last step of the classification scheme that we follow during this study: build a classification model that is capable of good generalization to compositions that have not been used for training the different composer categories.

For the classification we use machine learning algorithms. These mechanisms need a benchmark data set in order to create a model by a classifier. Then, this model is used to perform the classification of the elements that we want to analyse.

In this step we use *sklearn* that is a machine learning suite in Python [5]. This package includes all the tools and classifiers that we need to process the features and classification data of these study, and present the results.

In Figure 3.7 there is the blocks diagram of the last step of the classification scheme, including variables loading, runs composition, folds creation and classification. These parts will be explained in the following sections.

3.4.1 Variables loading

The variables loading is the main first step for the classification, so they will be used by the classifiers in order to perform the task. The fields that we need to know in these steps are: list of feature groups and minimum works threshold

[5] http://scikit-learn.org/stable/

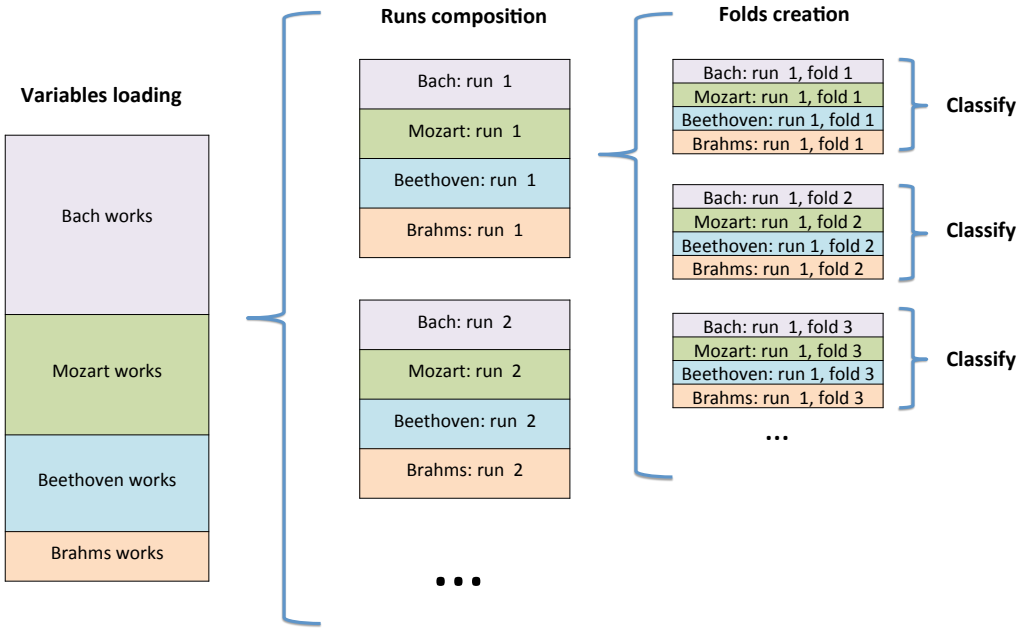

Figure 3.7: Blocks diagram of the last classification step

per composer requested by the experiment. Using these specifications, we load all the information that fits in both characteristics in order to use the maximum information from the database as possible.

In this step we have to construct a single vector per each work containing the values from all requested feature groups.

3.4.2 Runs composition

In this step we structure the runs of our classification. If we don't specify any run number in our experiments, the default number of runs in this study is 10. The goal to use different runs in our classifications experiments is to avoid good or bad results due to particular easy classification experiment; i.e. each run intends to be an individual classification experiment in order to achieve one accuracy per run, and then be able to perform the mean of all the runs and receive a proper and realistic result.

In the runs composition we follow the next steps:

1. Filtering of the features with variance 0; i.e. removing of all the features from all works that are the same value in all them.

2. Set the number of instances per class to classify: this number can be determined for fixed by the experiment and in any case equal to or smaller than

the number of works by the composer with less works.

3. Perform a shuffling among the works of the same composer. With this we avoid to have runs that contains similar music structures by the same composer. For instance, we avoid that all Beethoven sonatas, all Chopin prelude, etc., belong to the same run, dispersing them into separate runs randomly.

4. Each run is a table where each column corresponds to all feature value from all the works and, respectively, each row are all feature values of a musical work.

5. Create the runs taking into account several conditions in order to avoid te repetition of works as much as possible: In a run can not be any repeated work; We are able to repeat works (in different runs), if all the remaining works by the same composer are already in a run.

6. We have to create a vector alongside with contains an integer identifier for each composer. Each composer needs a single identifier for all the general classification experiment.

7. We have to repeat the steps 3, 4, 5 and 6 for each run requested taking into account the specified rules.

3.4.3 K-Fold cross validation

Once we have structured all the runs that we have to classify, we are able to perform the classification of these runs. We apply the K-fold cross validation model variation technique in each of these runs. If we don't specify any folds number in our experiments, the default number of folds is 10.

The K-Fold cross validation is used in order to be able to test all the instances from a classification experiment. The segmentation of each run in the respective folds is performed by a function of sklearn package called *stratified K fold*, which creates one K-fold from each run that we have created. In Figure 3.8 there is an example of 10-Fold cross validation.

3.4.4 Classifiers

The classification itself is the last step of the experiments. In this part we have to bring to classifiers each fold from the run in order to perform the global experiment (Number of runs * Number of Folds = Number of classifications).

Each fold contains a training set and a test set: the former is handled to create the model by the classifier, and the latter is classified using the created model;

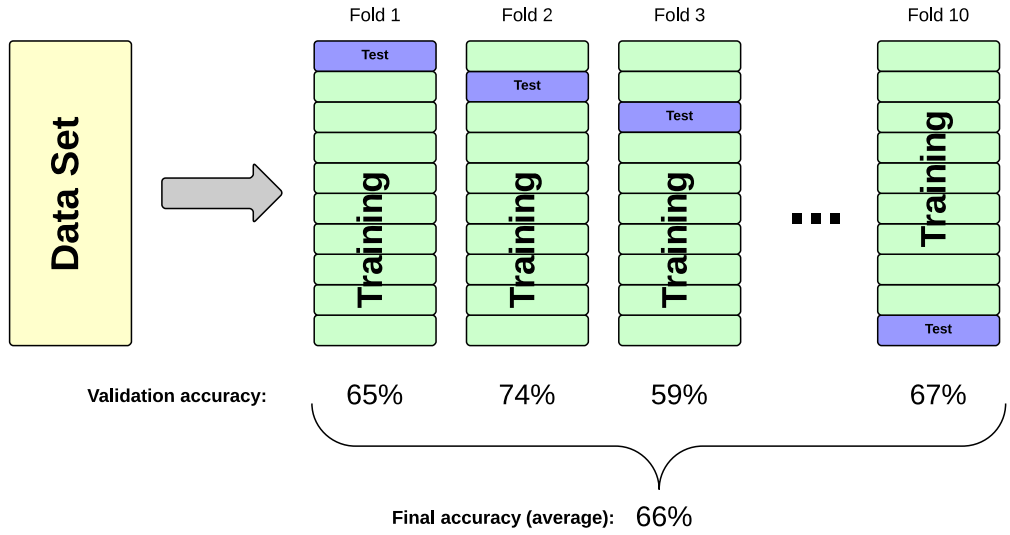

Figure 3.8: Example of data set performed to folds in 10-Fold Cross Validation

finally, the system calculates the accuracy percentage of the classification of the fold. With all the accuracies from all the classification experiments, we are able to compute the mean and the standard deviation of the global classification experiment.

In this study we use two of the most used classifiers in MIR tasks. These classifiers are Support Vector Machines (SVM) and Random Forest. We have chosen these classifiers because both perform a feature selection and they also return information of the weight coefficients that they uses in the classifications. These coefficients will be analysed in order to know which are the best features for each classifier. We also use a pseudo-classifier which give us a baseline of the classifications. All of them are provided by *sklearn* package.

Change baseline

In order to get a baseline of our classification experiments, we use a method called *Dummy* from the sklearn package. This method has implemented different strategies based on random classifications. Its classification accuracy is useful as a baseline in order to be compared with the following classifier accuracies.

In this study we use the *uniform* strategy of the classifier, which performs predictions uniformly at random. We use the default configuration in the remaining parameters.

Support Vector Machines

SVM classifier is a well know algorithm used in classification and regression problems. There are a few implementations of it. In this study we use the *linear SVM* which uses kernel linear and regularization. [25]

The SVM classification algorithm tries to find a hyperplane which is able to separate the d-dimensional data perfectly into its two different classes. However, since example data is often not linearly separable, this algorithm introduces the usage of kernels. These ones arrange the data in a multidimensional space where several hyperplanes have to separate the data. This classifier can deal with a high number of features, avoiding over-fitting, which happens when the number of features is much larger than the number of classified instances.

In this study we change some parameters from the default configuration: penalty = 'l1' (specifies the norm that has to be used in the penalization of features that are not relevant for the classification) and dual = False (sets the problem to the primal formulation). Using these parameters we will know the weight used per each feature in the classifications in order to use it in the feature selection.

Random Forest

Another well known algorithm is Random Forest based on decision trees algorithms, which used in classification and regression problems as well. In this study we use the implementation called *Random Forest Classifier*.

This algorithm starts creating a *n* bootstrap (estimators) from the original data. For each created bootstrap, it creates a single classification problem. Connecting these sub-classifications (called trees), the algorithm is able to perform a general classification depending of the decisions taken in each of these trees.

We set the parameter n_estimators = 20, which is related to the number of trees in the forest that the classifier uses in the model construction. The remaining parameters are set in the default values.

As SVM classifier, Random Forest also give us information about which is the weight that the classifier used for the feature selection in order to know which are the parameters that are better for the classification using this algorithm.

Chapter 4

RESULTS

This chapter includes the results achieved in the experiments proposed in this study using the methodology already explained. We also will expose the feature selection performance, followed by extensive results analysis.

4.1 Preliminary results

Our first experiments consist on the classification of the different groups of composers using separately each feature group in order to know which are the accuracies that are able to achieve.

4.1.1 Accuracies per feature groups

In Tables 4.1, 4.2 and 4.3 there are the average results and the standard deviations for each feature group of the classification over the composer groups exposed in Section 3.1.2.

In all the classification groups, the best results are achieved by the Random Forest classifier. For all the classifications, the best feature is the *Differential* with threshold of 0. This feature is related to the differential over consecutive chroma beats, which means how different are the chords from these beats: the higher are the feature values, the more different are chroma classes between consecutive beats; the lower are the feature values, the more similar are chroma classes between these beats.

The second best classifier group is the same for SVM classifications: the other Differential feature with threshold of 0.5 gets the second place. In the RF classifier the second place is achieved by *Chromagram Histogram* feature in the case of more than 100 works, whereas *Combinations Histogram* feature with threshold of 0.5 gets the second place in the remaining cases. However, the accuracy difference

Feature groups		Dummy	Random Forest	SVM
Feature type	**Parameters**	**Mean (std) [%]**	**Mean (std) [%]**	**Mean (std) [%]**
binary_vec	-	0.9 (0.6)	18.7 (4.2)	7.4 (1.4)
histogram_vec	-	1.0 (0.6)	18.8 (4.8)	7.1 (1.3)
empty_hist_bins	-	0.9 (0.6)	3.9 (1.1)	0.9 (0.2)
profile_template_correlation	-	0.9 (0.6)	9.9 (4.0)	1.7 (0.5)
internal_correlation	-	0.9 (0.6)	19.0 (4.1)	9.1 (1.6)
differential	Thr. = 0	0.9 (0.7)	**21.5 (4.5)**	**16.9 (2.2)**
	Thr. = 0.5	0.9 (0.0)	20.1 (4.2)	**15.8 (2.4)**
sumatory_columns	-	0.9 (0.0)	16.9 (4.2)	7.2 (1.6)
beat_pres_chroma	Thr. = 0.5	0.9 (0.7)	17.3 (4.4)	6.6 (1.2)
	Thr. = 1	1.0 (0.7)	15.6 (4.1)	4.3 (1.0)
comb	Thr. = 1	0.9 (0.0)	18.4 (4.4)	5.0 (1.2)
	Thr. = 0.5	0.9 (0.6)	**20.6 (4.2)**	7.7 (1.3)
interval_unison_val	-	1.0 (0.6)	17.1 (4.4)	8.1 (1.4)
rel_interval_val	-	0.9 (0.6)	16.1 (4.1)	5.1 (1.1)
relation_chord	Cl. = 6, Lg = 1, Thr. = 0.5	0.9 (0.6)	15.3 (4.5)	3.4 (0.8)
	Cl. = 6, Lg = 2, Thr. = 0.5	0.9 (0.6)	15.4 (4.4)	3.6 (1.0)
	Cl. = 6, Lg = 3, Thr. = 0.5	0.9 (0.7)	15.3 (4.6)	3.8 (0.9)
	Cl. = 6, Lg = 4, Thr. = 0.5	0.9 (0.0)	15.2 (4.2)	3.9 (1.0)
	Cl. = 13, Lg = 1, Thr. = 0.5	0.9 (0.0)	19.7 (4.2)	6.9 (1.3)
	Cl. = 13, Lg = 2, Thr. = 0.5	0.9 (0.7)	19.2 (4.2)	7.1 (1.2)
	Cl. = 13, Lg = 3, Thr. = 0.5	1.0 (0.7)	18.9 (4.4)	7.4 (1.3)
	Cl. = 6, Lg = 1, Thr. = 0	0.9 (0.0)	12.4 (3.8)	2.0 (0.6)
	Cl. = 6, Lg = 2, Thr. = 0	0.9 (0.0)	13.1 (3.8)	2.1 (0.6)
	Cl. = 6, Lg = 3, Thr. = 0	0.9 (0.0)	12.6 (3.8)	2.2 (0.7)
	Cl. = 6, Lg = 4, Thr. = 0	0.9 (0.0)	12.8 (4.0)	2.3 (0.7)
	Cl. = 13, Lg = 1, Thr. = 0	0.9 (0.0)	16.7 (4.6)	4.5 (1.0)
	Cl. = 13, Lg = 2, Thr. = 0	0.9 (0.0)	17.0 (4.1)	5.0 (1.2)
	Cl. = 13, Lg = 3, Thr. = 0	0.9 (0.7)	16.2 (4.2)	5.0 (1.0)
chord_simp	Cl. = 6, Thr. = 0.5	0.9 (0.0)	13.8 (4.0)	3.3 (0.8)
	Cl. =13, Thr. = 0.5	0.9 (0.0)	19.9 (3.7)	7.6 (1.2)
	Cl. = 6, Thr. = 0	0.9 (0.7)	10.5 (3.4)	2.0 (0.7)
	Cl. =13, Thr. = 0	1.0 (0.7)	16.2 (4.1)	3.9 (0.9)
chord_shift	Lg. = 0, Thr. = 0.5, No Trans.	0.9 (0.0)	18.8 (4.0)	6.4 (1.1)
	Lg. = 1, Thr. = 0.5, No Trans.	0.9 (0.0)	18.7 (4.1)	7.0 (1.1)
	Lg. = 2, Thr. = 0.5, No Trans.	0.9 (0.0)	17.7 (4.3)	7.4 (1.4)
	Lg. = 0, Thr. = 0.5, Trans.	0.9 (0.0)	16.3 (3.7)	4.9 (1.0)
	Lg. = 1, Thr. = 0.5, Trans.	0.9 (0.0)	17.0 (4.4)	5.7 (1.3)
	Lg. = 2, Thr. = 0.5, Trans.	0.9 (0.0)	17.0 (4.1)	5.9 (1.5)
	Lg. = 0, Thr. = 0, No Trans.	1.0 (0.7)	15.0 (4.0)	3.6 (0.9)
	Lg. = 1, Thr. = 0, No Trans.	1.0 (0.6)	15.6 (4.2)	3.9 (0.9)
	Lg. = 2, Thr. = 0, No Trans.	1.0 (0.7)	15.5 (4.3)	4.0 (1.0)
	Lg. = 0, Thr. = 0, Trans.	0.8 (0.7)	14.1 (4.5)	3.2 (0.9)
	Lg. = 1, Thr. = 0, Trans.	0.9 (0.7)	15.1 (4.2)	3.5 (1.1)
	Lg. = 2, Thr. = 0, Trans.	0.8 (0.7)	14.8 (4.3)	3.6 (1.0)
hist_transposed_chord	Cl. = 13, Lg = 1, Thr. = 0.5	0.9 (0.6)	18.7 (4.7)	6.4 (1.1)
	Cl. = 13, Lg = 2, Thr. = 0.5	1.0 (0.6)	17.7 (4.0)	6.8 (1.4)
	Cl. = 6, Lg = 1, Thr. = 0.5	1.0 (0.7)	15.1 (4.3)	2.8 (0.6)
	Cl. = 6, Lg = 2, Thr. = 0.5	0.9 (0.7)	15.4 (4.4)	2.9 (0.7)
	Cl. = 13, Lg = 1, Thr. = 0	1.1 (0.7)	16.4 (4.3)	3.1 (0.8)
	Cl. = 13, Lg = 2, Thr. = 0	1.0 (0.7)	16.5 (4.4)	3.6 (0.8)
	Cl. = 6, Lg = 1, Thr. = 0	0.9 (0.7)	11.2 (3.3)	1.6 (0.6)
	Cl. = 6, Lg = 2, Thr. = 0	1.0 (0.6)	12.8 (3.8)	1.8 (0.8)

Table 4.1: Feature groups accuracies for composers with more than 20 works

Feature groups		Dummy	Random Forest	SVM
Feature type	**Parameters**	**Mean (std) [%]**	**Mean (std) [%]**	**Mean (std) [%]**
binary_vec	-	2.1 (0.0)	29.0 (4.9)	13.0 (1.8)
histogram_vec	-	2.1 (0.8)	29.5 (4.9)	15.9 (2.1)
empty_hist_bins	-	2.1 (0.9)	7.3 (1.6)	2.7 (0.5)
profile_template_correlation	-	2.1 (0.0)	14.4 (5.1)	3.5 (0.7)
internal_correlation	-	2.1 (0.0)	28.1 (5.2)	17.1 (2.3)
differential	Thr. = 0	2.1 (0.0)	**31.9 (4.5)**	**23.0 (2.5)**
	Thr. = 0.5	2.1 (0.0)	30.6 (4.4)	**20.0 (2.4)**
sumatory_columns	-	2.1 (0.0)	24.5 (4.7)	11.8 (1.7)
beat_pres_chroma	Thr. = 0.5	2.1 (0.0)	26.2 (4.6)	12.1 (1.5)
	Thr. = 1	2.2 (1.0)	23.2 (5.2)	8.5 (1.3)
comb	Thr. = 1	2.1 (1.0)	27.0 (4.8)	9.8 (1.5)
	Thr. = 0.5	2.1 (1.0)	**30.8 (4.9)**	16.2 (1.8)
interval_unison_val	-	2.1 (1.0)	26.7 (4.7)	17.2 (1.8)
rel_interval_val	-	2.2 (0.9)	24.9 (4.8)	10.1 (1.6)
relation_chord	Cl. = 6, Lg = 1, Thr. = 0.5	2.1 (1.1)	23.5 (4.6)	8.7 (1.6)
	Cl. = 6, Lg = 2, Thr. = 0.5	2.1 (0.0)	23.5 (4.7)	8.7 (1.5)
	Cl. = 6, Lg = 3, Thr. = 0.5	2.1 (0.8)	23.6 (4.7)	9.0 (1.5)
	Cl. = 6, Lg = 4, Thr. = 0.5	2.1 (0.9)	23.6 (5.2)	8.8 (1.4)
	Cl. = 13, Lg = 1, Thr. = 0.5	2.1 (0.0)	30.5 (4.4)	15.9 (1.8)
	Cl. = 13, Lg = 2, Thr. = 0.5	2.1 (0.0)	29.9 (5.0)	16.7 (1.7)
	Cl. = 13, Lg = 3, Thr. = 0.5	2.1 (0.0)	28.7 (4.7)	16.6 (2.1)
	Cl. = 6, Lg = 1, Thr. = 0	2.1 (0.0)	19.4 (4.9)	4.0 (0.9)
	Cl. = 6, Lg = 2, Thr. = 0	2.1 (0.0)	19.6 (4.7)	4.5 (0.9)
	Cl. = 6, Lg = 3, Thr. = 0	2.1 (0.0)	20.1 (5.0)	4.7 (0.8)
	Cl. = 6, Lg = 4, Thr. = 0	2.2 (1.0)	19.9 (4.3)	4.9 (0.8)
	Cl. = 13, Lg = 1, Thr. = 0	2.1 (1.0)	26.5 (4.8)	9.8 (1.5)
	Cl. = 13, Lg = 2, Thr. = 0	2.1 (1.0)	26.5 (5.0)	10.3 (1.5)
	Cl. = 13, Lg = 3, Thr. = 0	2.1 (1.0)	26.7 (4.7)	10.5 (1.6)
chord_simp	Cl. = 6, Thr. = 0.5	2.2 (0.9)	21.1 (4.9)	7.4 (1.2)
	Cl. =13, Thr. = 0.5	2.1 (1.1)	29.9 (5.3)	16.3 (1.9)
	Cl. = 6, Thr. = 0	1.9 (0.8)	15.7 (4.1)	3.3 (0.8)
	Cl. =13, Thr. = 0	2.1 (0.9)	24.5 (4.9)	7.7 (1.4)
chord_shift	Lg. = 0, Thr. = 0.5, No Trans.	2.2 (1.0)	27.5 (4.9)	10.9 (1.5)
	Lg. = 1, Thr. = 0.5, No Trans.	2.3 (1.1)	27.9 (5.2)	13.5 (1.9)
	Lg. = 2, Thr. = 0.5, No Trans.	2.3 (0.9)	26.8 (4.7)	13.7 (1.8)
	Lg. = 0, Thr. = 0.5, Trans.	2.1 (0.0)	26.0 (4.6)	9.7 (1.2)
	Lg. = 1, Thr. = 0.5, Trans.	2.1 (0.0)	27.7 (5.3)	12.9 (1.9)
	Lg. = 2, Thr. = 0.5, Trans.	2.1 (0.0)	26.8 (4.7)	13.2 (1.7)
	Lg. = 0, Thr. = 0, No Trans.	2.2 (1.0)	22.9 (4.7)	6.9 (1.1)
	Lg. = 1, Thr. = 0, No Trans.	2.1 (1.0)	23.9 (4.9)	8.0 (1.4)
	Lg. = 2, Thr. = 0, No Trans.	2.1 (1.0)	23.5 (5.0)	8.1 (1.4)
	Lg. = 0, Thr. = 0, Trans.	2.1 (1.0)	22.4 (5.0)	6.0 (1.0)
	Lg. = 1, Thr. = 0, Trans.	2.2 (0.9)	23.4 (5.1)	7.1 (1.4)
	Lg. = 2, Thr. = 0, Trans.	2.1 (1.1)	23.2 (4.7)	7.4 (1.2)
hist_transposed_chord	Cl. = 13, Lg = 1, Thr. = 0.5	1.9 (0.8)	30.0 (4.7)	13.3 (1.7)
	Cl. = 13, Lg = 2, Thr. = 0.5	2.1 (0.9)	27.5 (4.8)	14.8 (1.8)
	Cl. = 6, Lg = 1, Thr. = 0.5	2.2 (1.0)	24.7 (4.9)	5.7 (1.1)
	Cl. = 6, Lg = 2, Thr. = 0.5	2.3 (1.1)	25.0 (4.9)	6.1 (1.2)
	Cl. = 13, Lg = 1, Thr. = 0	2.3 (0.9)	25.5 (5.0)	7.0 (1.2)
	Cl. = 13, Lg = 2, Thr. = 0	2.2 (0.9)	25.1 (4.7)	7.6 (1.3)
	Cl. = 6, Lg = 1, Thr. = 0	2.1 (0.9)	17.4 (4.5)	3.3 (0.6)
	Cl. = 6, Lg = 2, Thr. = 0	2.1 (1.0)	20.3 (4.6)	3.5 (0.8)

Table 4.2: Feature groups accuracies for composers with more than 50 works

Feature groups		Dummy	Random Forest	SVM
Feature type	**Parameters**	**Mean (std) [%]**	**Mean (std) [%]**	**Mean (std) [%]**
binary_vec	-	5.6 (1.7)	34.4 (4.8)	23.4 (2.7)
histogram_vec	-	5.9 (1.8)	**36.7 (4.6)**	27.9 (3.0)
empty_hist_bins	-	5.8 (1.7)	12.3 (2.6)	8.8 (1.4)
profile_template_correlation	-	5.8 (1.8)	13.8 (4.5)	8.8 (1.2)
internal_correlation	-	5.8 (1.6)	33.4 (4.1)	25.2 (2.9)
differential	Thr. = 0	5.8 (1.8)	**37.9 (4.8)**	**29.9 (3.2)**
	Thr. = 0.5	5.8 (1.6)	35.9 (4.4)	**28.1 (2.6)**
sumatory_columns	-	5.6 (1.6)	28.2 (4.5)	17.2 (2.3)
beat_pres_chroma	Thr. = 0.5	5.9 (0.0)	29.2 (4.5)	18.3 (2.1)
	Thr. = 1	5.6 (1.7)	26.0 (4.3)	15.7 (2.4)
comb	Thr. = 1	5.9 (1.8)	31.7 (4.1)	17.5 (2.3)
	Thr. = 0.5	5.8 (1.7)	35.6 (4.4)	24.6 (2.3)
interval_unison_val	-	5.8 (1.8)	30.8 (4.7)	25.8 (2.6)
rel_interval_val	-	5.8 (1.6)	28.4 (4.0)	16.5 (2.2)
relation_chord	Cl. = 6, Lg = 1, Thr. = 0.5	5.9 (1.8)	27.3 (4.5)	19.2 (2.5)
	Cl. = 6, Lg = 2, Thr. = 0.5	5.8 (1.7)	27.9 (4.1)	18.7 (2.1)
	Cl. = 6, Lg = 3, Thr. = 0.5	5.8 (1.8)	27.7 (4.5)	19.3 (2.6)
	Cl. = 6, Lg = 4, Thr. = 0.5	5.8 (1.6)	27.8 (4.9)	18.9 (2.5)
	Cl. = 13, Lg = 1, Thr. = 0.5	5.9 (1.8)	34.7 (4.8)	25.1 (2.5)
	Cl. = 13, Lg = 2, Thr. = 0.5	5.8 (1.7)	32.9 (4.7)	25.8 (2.6)
	Cl. = 13, Lg = 3, Thr. = 0.5	5.8 (1.8)	31.8 (4.8)	25.8 (2.3)
	Cl. = 6, Lg = 1, Thr. = 0	5.8 (1.6)	18.2 (3.9)	9.9 (1.6)
	Cl. = 6, Lg = 2, Thr. = 0	5.8 (1.7)	18.5 (4.1)	9.8 (1.4)
	Cl. = 6, Lg = 3, Thr. = 0	6.0 (1.6)	19.1 (3.9)	9.7 (1.7)
	Cl. = 6, Lg = 4, Thr. = 0	5.8 (1.7)	19.3 (4.0)	10.1 (1.8)
	Cl. = 13, Lg = 1, Thr. = 0	5.9 (1.8)	31.4 (4.3)	19.7 (2.0)
	Cl. = 13, Lg = 2, Thr. = 0	5.8 (1.7)	31.3 (3.9)	19.8 (2.0)
	Cl. = 13, Lg = 3, Thr. = 0	5.8 (1.8)	32.3 (4.8)	20.3 (2.1)
chord_simp	Cl. = 6, Thr. = 0.5	5.8 (1.6)	25.0 (4.6)	17.2 (1.6)
	Cl. =13, Thr. = 0.5	5.8 (1.7)	34.3 (4.9)	24.5 (2.8)
	Cl. = 6, Thr. = 0	6.0 (1.6)	15.9 (3.6)	9.8 (1.4)
	Cl. =13, Thr. = 0	5.8 (1.7)	29.3 (4.3)	17.5 (2.2)
chord_shift	Lg. = 0, Thr. = 0.5, No Trans.	5.9 (0.0)	31.2 (4.0)	16.6 (2.2)
	Lg. = 1, Thr. = 0.5, No Trans.	5.9 (0.0)	32.1 (4.9)	21.4 (2.5)
	Lg. = 2, Thr. = 0.5, No Trans.	5.9 (0.0)	31.5 (4.9)	22.0 (2.3)
	Lg. = 0, Thr. = 0.5, Trans.	5.9 (0.0)	30.0 (4.4)	16.0 (2.3)
	Lg. = 1, Thr. = 0.5, Trans.	5.9 (0.0)	32.2 (4.6)	22.7 (2.6)
	Lg. = 2, Thr. = 0.5, Trans.	5.9 (0.0)	31.7 (4.4)	23.0 (2.5)
	Lg. = 0, Thr. = 0, No Trans.	5.9 (1.7)	27.3 (4.5)	15.7 (2.2)
	Lg. = 1, Thr. = 0, No Trans.	6.1 (1.8)	29.3 (4.3)	17.1 (2.3)
	Lg. = 2, Thr. = 0, No Trans.	6.1 (1.7)	28.6 (4.5)	17.1 (2.3)
	Lg. = 0, Thr. = 0, Trans.	5.6 (1.7)	25.4 (4.3)	12.4 (1.9)
	Lg. = 1, Thr. = 0, Trans.	5.9 (1.8)	28.1 (4.5)	15.2 (2.1)
	Lg. = 2, Thr. = 0, Trans.	5.8 (1.7)	28.3 (4.4)	14.8 (2.2)
hist_transposed_chord	Cl. = 13, Lg = 1, Thr. = 0.5	5.8 (1.8)	35.8 (4.7)	23.7 (3.0)
	Cl. = 13, Lg = 2, Thr. = 0.5	5.8 (1.6)	33.8 (4.8)	25.3 (2.9)
	Cl. = 6, Lg = 1, Thr. = 0.5	5.8 (1.7)	31.6 (4.5)	14.1 (2.0)
	Cl. = 6, Lg = 2, Thr. = 0.5	6.0 (1.6)	30.8 (5.0)	15.4 (2.4)
	Cl. = 13, Lg = 1, Thr. = 0	5.8 (1.7)	30.6 (4.4)	14.5 (1.9)
	Cl. = 13, Lg = 2, Thr. = 0	5.8 (1.8)	31.3 (4.4)	15.9 (2.2)
	Cl. = 6, Lg = 1, Thr. = 0	6.0 (1.8)	20.7 (3.5)	9.6 (1.2)
	Cl. = 6, Lg = 2, Thr. = 0	6.2 (1.8)	21.9 (4.2)	9.0 (1.4)

Table 4.3: Feature groups accuracies for composers with more than 100 works

between the top ten features is less than 5% meaning that even the information is related to different concepts, these concepts have the same importance in terms of identify the works' composers.

The consistence of the results tell us that the features are designed correctly, but we are interested in studying if their combination yields better results.

4.1.2 Feature selection

In this section we will expose the procedure to find combinations of feature groups in order to achieve the best possible results.

For the feature selection, we use the composers group of 50 works, because it is in the middle of the remaining composers groups. However, we need to split this group in two half parts: one part will be our test set and the other the validation set. The splitting has to be stratified, because we need the same number of works per composer in each data set (or almost the same, for the cases of composers with even number of works).

The feature selection experiments will be performed using the test set. The validation set only will be used at the end of the feature selection in order to check if the results are consistent.

Due to time constraints we are not able to perform all the feature combinations that we would want in this study. Is because of that that we make selection of the features manually, but based on the logic and intuition based on the results analysis.

The starting point of the feature selection is to find the best 10 feature groups from the classification. In Tables 4.4 and 4.5 there are the top 10 ranking of the best feature groups classifications.

In all the classification experiments we always include the feature groups with less than 10 features. These features are:

- Empty bins histogram (1 feature)
- Profile template correlation number (1 feature)
- Sumatory columns (6 features)
- Chord simplification with threshold of 0 (6 features)
- Chord simplification with threshold of 0.5 (6 features)

The feature selection is performed separately for each classifier because they use different procedures to perform classification experiments, so they may need different features in order to achieve the best results as possible.

Feature groups		Random Forest	Ranking
Feature type	**Parameters**	**Mean (std) [%]**	**position**
differential	Thr. = 0	31.9 (4.5)	1
comb	Thr. = 0.5	30.8 (4.9)	2
differential	Thr. = 0.5	30.6 (4.4)	3
relation_chord	Cl. = 13, Lg = 1, Thr. = 0.5	30.5 (4.4)	4
hist_transposed_chord	Cl. = 13, Lg = 1, Thr. = 0.5	30.0 (4.7)	5
relation_chord	Cl. = 13, Lg = 2, Thr. = 0.5	29.9 (5.0)	6
chord_simp	Cl. =13, Thr. = 0.5	29.9 (5.3)	7
histogram_vec	-	29.5 (4.9)	8
binary_vec	-	29.0 (4.9)	9
relation_chord	Cl. = 13, Lg = 3, Thr. = 0.5	28.7 (4.7)	10

Table 4.4: Top 10 ranking of feature groups of the composers group with minimum of 50 works, using the RF classifier

Feature groups		SVM	Ranking
Feature type	**Parameters**	**Mean (std) [%]**	**position**
differential	Thr. = 0	23.0 (2.5)	1
differential	Thr. = 0.5	20.0 (2.4)	2
interval_unison_val	-	17.2 (1.8)	3
internal_correlation	-	17.1 (2.3)	4
relation_chord	Cl. = 13, Lg = 2, Thr. = 0.5	16.7 (1.7)	5
relation_chord	Cl. = 13, Lg = 3, Thr. = 0.5	16.6 (2.1)	6
chord_simp	Cl. =13, Thr. = 0.5	16.3 (1.9)	7
comb	Thr. = 0.5	16.2 (1.8)	8
histogram_vec	-	15.9 (2.1)	9
relation_chord	Cl. = 13, Lg = 1, Thr. = 0.5	15.9 (1.8)	10

Table 4.5: Top 10 ranking of feature groups of the composers group with minimum of 50 works, using the SVM classifier

In Random Forest classifier, we start the feature selection with another classification of the top 10 feature groups but adding the fixed features that we commented in the previous enumeration. We also repeat these experiment because the database have been reduced to the half, so the results may have changed due to both fact. In the first block of the Figure 4.1 there are the results of these classifications. The results are different around a 2% in most of the cases if we compare with the results of the 50 works threshold.

The next step is to make several combinations of two elements using the best results of the first block. The accuracies of these combinations are presented in the second block of the Figure 4.1, where they are slightly better (less than 1%).

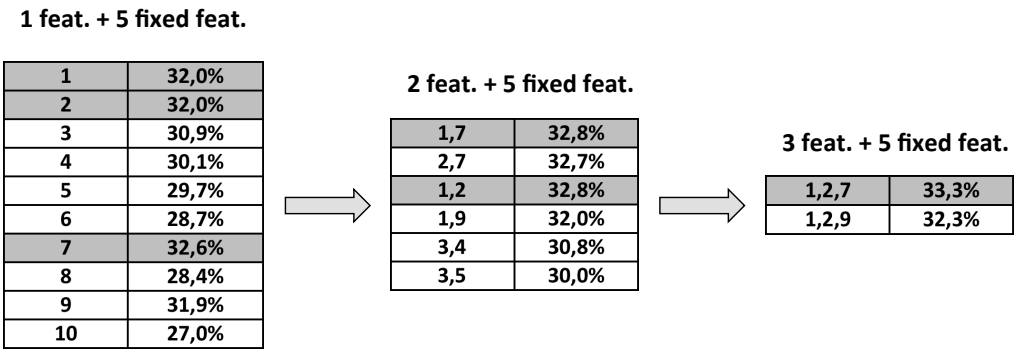

Figure 4.1: Principal group features combinations using Random Forest Classifier. For each classification there is the combination identification (using the numbers corresponding to the ranking position in Table 4.4) and its classification accuracy.

Then, we perform another combination based on the 2nd block results. The results achieved are better than the accuracies in the 2n block, but these differences still not much relevants (less than 1%). We try combinations using more elements as well, even we create classifications merging 10 and 9 feature groups, but all of these combinations are worse than the one achieved by 1,2 and 7.

In order to try to improve the classification, we also try to add to the combination more feature groups with low number of features, which could increase the accuracy of the selected combination. The chosen features in this experiment belongs to different types of feature groups in order to provide other information no present in the top ones.

The best accuracy achieved is 33,9% when we add two features to the previous combination: Chromagram Histogram and Chords Relation using 6 classes, 1 lag and threshold of 0. However, the accuracy is only 0.6% higher than the classification performed without these features.

The last step for the feature selection is to validate the accuracies achieved using the validation set. We perform the classification to the top five combinations

of selected features. The results are consistent according to the test set: the best combinations for the test set achieves an accuracy of 35,2%, the second place in the validation set, just 0.05 % lower than the best result for the validation set.

The feature groups included in the final combination for the Random Forest classifier are:

- Differential with threshold of 0
- Combinations Histograms with threshold of 0.5
- Chord Simplifications using 13 classes and threshold of 0.5
- Chromagram Histogram
- Chords Relation using 6 classes, 1 lag and threshold of 0

In this combination there is information related to several aspects of music scores: the relation between beats is located in differentials (numeric) and chords relation (type of chords), the chromagram histogram provides information about the exactly chroma combinations over all the work, and the chord simplification and the combinations histograms are the distributions of the chords over the work (the first with simplification of 13 classes of chords and the second with the explicit type of chord).

In the feature selection for the SVM classifier, we start with the same approach than in the previous classifier: repeating the classifications of the top 10 features for the test set. In this case, the ranking is practically the same than in the classification of 50 works threshold, just exchanging the positions of 3rd and 4th feature groups. The achieved results have been improved as well around a 2-3% in all the feature groups. In the first block of Figure 4.2 there are the results of this part.

Then, we create combinations of two feature groups using the features with best results from the first block plus the fixed feature groups. The accuracies are presented in the second block of the same Figure, with an improving of 2% comparing to the best result from the first block. However, then we decided to increase the number of feature groups for each experiment, like is showed in the 3rd block of Figure 4.2. Using the best combination from this block, we tried with other feature groups from the top 10 ranking. The final combination is made with 6 of the features from the ranking.

Finally, as we have done in the other classifier, we included other features to the combination that we think that could contribute to the classification because of their musical information. We included Chroma Vector feature to the classification, which improves the classification achieving a 30%.

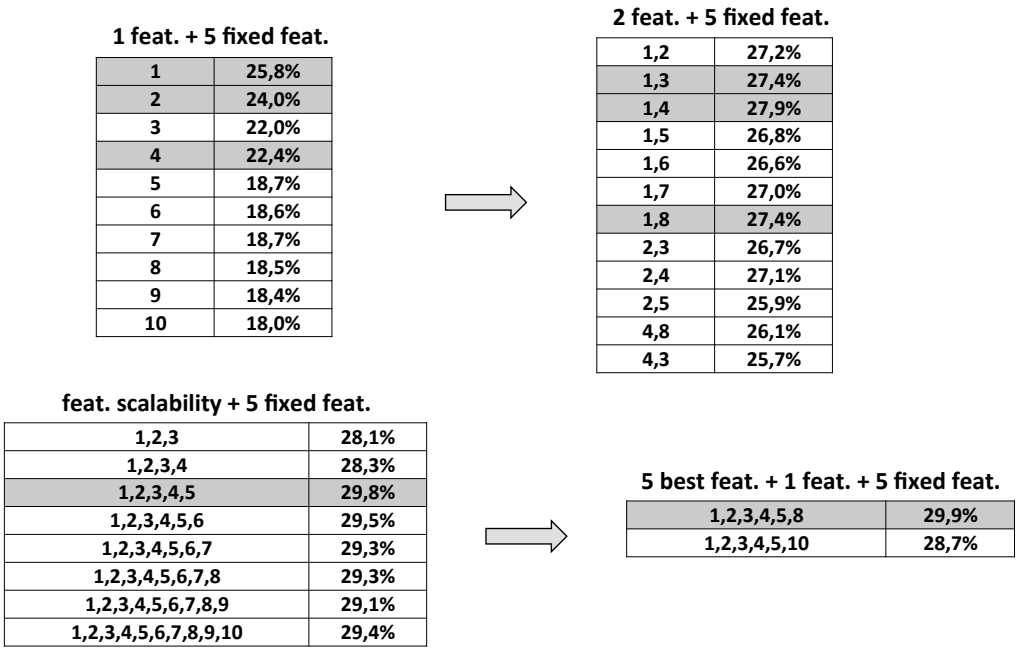

Figure 4.2: Principal group features combinations using Random Forest Classifier. For each classification there is the combination identification (using the numbers corresponding to the ranking position in Table 4.5) and its classification accuracy.

We perform the classification of the best combinations using the validation set. The selected feature is in the 4th position with an accuracy 0.3% lower than in the test set, so this proves the consistence of the combination.

The feature groups included in the final combination for the SVM classifier are:

- Differential with threshold of 0
- Differential with threshold of 0.5
- Intervals between strongest unisons
- Internal Correlation
- Chord Relation using 13 classes, 2 lags and threshold of 0.5
- Combinations Histogram
- Chroma Vector

This combination contains information related to several aspects to music scores as well: both differential features that give use information about the differences between with two levels of notes importance, intervals between strongest unisons that tries to follow the melody or lead voice of the score, the internal correlation is another way to extract how similar are the consecutive beats, the chord relation and the combinations histogram perform two different distributions of the chords of the whole score and the chroma vector creates an overall view of each chroma class.

In Table 4.1.2 we present the feature combinations for each classifier that will be used in the following sections (we have to include the fixed features in both lists). However, we have to take into account that these feature combinations are selected in order to be used whit a works threshold of 25 works (because we were using the half of the dataset of works threshold of 50). It means that other combinations could achieve better results using another number of works for the classification experiments. In both combinations we include Differential, Chromagram Histogram and Chord Relation features with different parameters.

Random Forest	**SVM**
Differential with threshold of 0	Differential with threshold of 0
Combinations Histograms with threshold of 0.5	Differential with threshold of 0.5
Chord Simplifications using 13 classes and threshold of 0.5	Intervals between strongest unisons
Chromagram Histogram	Chromagram Histogram
Chords Relation using 6 classes, 1 lag and threshold of 0	Chord Relation using 13 classes, 2 lags and threshold of 0.5
	Internal Correlation
	Chroma Vector

Table 4.6: Best feature combinations for both classifiers

4.2 Final results

4.2.1 Groups accuracies

In Table 4.7 we expose the final accuracies, using the combination resulting of the feature selection, for groups of the data set created depending of the minimum number of works per composer.

Classifier /Works threshold	20 Works Mean (std) [%]	50 Works Mean (std) [%]	100 Works Mean (std) [%]
Random Forest	**27,6 (3,6)**	**39,0 (4,6)**	44,1 (3,8)
SVM	24,6 (3,2)	37,3 (3,6)	**46,7 (3,8)**
Baseline	1,0 (0,6)	2,3 (0,9)	5,5 (1,6)

Table 4.7: Final groups accuracies using feature selection. Best results are highlighted.

The composer group with a threshold of 20 works is the one that achieves better relative results compared with the baseline: 27,1 times the baseline classification using Random Forest classifier. In addition, if we use the feature groups combination instead of singles feature groups, we achieve increments up to +16,8% in the case of the composer groups with a works threshold of 100 using the SVM classifier. However, this improvement is not as much in other cases, for instance, in the classification of composers with more than 20 works with Random Forest classifier where the increase respect the baseline is +6,1%.

4.2.2 Confusion matrix

The confusion matrix is a display method that give us information about how a classification experiment distributed the instances to the respective classes comparing to the ground truth of the distribution; i.e. we are able to identify the most and the less conflictive classes as well.

The table disposition consist on the actual classes in rows and the predicted classes in columns; the diagonal values represent the good classified instances, the other values are from the wrong ones. The numbers from the cells are represented normally in % of the total instances of the actual class, so if we sum all the values from a row, it should give us 100%.

In Tables 4.8 and 4.9 we find the confusion matrices of the classification experiments using composer with more than 100 works, using Random Forest and SVM classifier respectively. In both tables, the most confused composers are located in the central part of the table, which belongs to composers of the classicism (around the year 1740). In the composers of the romanticism there are some confusion as well.

The best classified composers are from the Renaissance, where T.L. de Victoria achieves 91% of accuracy using the Random Forest classifier. This is because the rhythm of the Renaissance music is more static, regular and monotone that the rhythm from the other musical periods, where the rhythm is more complex and variant. On the other hand, the bad accuracies in the cases of L. van Beethoven or G. F. Händel, could be because their music belongs to the edge between two

different music periods (Baroque/Classical and Classical/Romanticism, respectively), so their music and techniques could be used for later composers in their works, inducing confusions between them.

In Tables 4.10 and 4.11 we present the confusion matrix that belongs to the experiments using Random Forest and SVM classifiers, achieved in the classification of composers with a minimum number of 20 works. In this case, due to space constraints, we only present the matrix confusion using the musical period of the composers. In these experiments, the best classification is achieved by composers from the Baroque period with 72,3% and 71,5% of accuracy. However, is the musical period that contains more wrong attributions of works from other periods, having a percentage of up to 41,3% of instances from other assigned to it. On the other hand, these confusion matrix indicates that the performance of the classification experiments by musical periods would achieve better results than the composer identification due to the reduction of the number of classes.

1543	Byrd, William	83,7	6,5	1,0	0,5	2,5	3,3	0,8	0,7	0,3	0,6	0,0	0,1	0,0	0,0	0,0	0,0	0,0
1548	Victoria, Tomas Luis de	3,4	91,0	2,0	0,3	0,3	0,3	0,6	0,6	0,3	0,3	0,6	0,2	0,0	0,1	0,1	0,0	0,0
1632	Lully, JeanBaptiste	3,9	5,1	64,1	1,8	3,9	6,0	2,8	6,2	1,5	1,6	1,7	0,3	0,0	0,2	0,5	0,1	0,2
1660	Scarlatti, Alessandro	1,5	0,6	1,0	50,6	2,4	4,7	8,5	3,9	6,3	5,1	2,8	2,5	1,1	2,7	1,5	2,8	2,2
1668	Couperin, Francois Le Grand	3,2	1,4	3,2	5,3	57,5	8,1	8,7	6,1	2,8	1,2	0,6	0,2	0,1	0,2	0,6	0,6	0,3
1682	Dandrieu, JeanFrancois	2,6	3,9	4,7	7,6	8,8	51,1	7,1	5,0	5,2	1,9	0,7	0,2	0,0	0,5	0,2	0,6	0,0
1685	Bach, Johann Sebastian	2,9	2,2	3,0	9,2	8,3	7,2	42,1	7,2	4,5	3,2	1,7	1,7	0,8	1,6	2,2	1,0	1,0
1685	Handel, Friedrich Georg	3,3	2,2	7,8	7,6	6,3	5,0	16,3	23,6	9,7	6,1	2,7	2,0	0,9	1,2	1,4	1,9	1,8
1732	Haydn, Franz Joseph	0,9	1,5	1,7	7,6	2,6	4,2	6,4	6,7	26,8	16,6	8,3	3,9	2,2	2,8	3,2	2,9	1,6
1756	Mozart, Wolfgang Amadeus	1,4	2,2	2,3	6,8	2,8	1,9	4,9	5,6	19,2	30,5	8,3	3,8	1,2	1,4	3,5	2,8	1,6
1770	Beethoven, Ludwig van	1,1	1,8	6,1	6,1	1,9	3,1	3,5	2,8	10,1	13,7	23,8	10,1	4,6	2,7	3,0	2,1	3,7
1797	Schubert, Franz Peter	0,0	0,3	0,6	2,7	0,0	0,3	2,3	0,9	6,2	5,3	11,7	30,6	8,2	6,5	7,6	7,2	9,5
1811	Liszt, Franz	0,1	0,8	0,8	3,5	0,2	0,8	2,2	2,0	4,6	2,3	3,8	12,7	39,1	8,6	4,9	7,5	6,1
1813	Alkan, Charles Henri Valentin	0,5	0,6	0,7	7,7	1,1	2,4	6,1	3,0	5,8	4,9	5,0	10,2	12,1	18,5	6,9	9,2	5,2
1833	Brahms, Johannes	0,2	0,1	1,2	2,7	0,9	2,2	5,3	3,5	3,3	4,2	3,3	9,9	4,0	3,8	43,2	4,1	8,1
1840	Tchaikovsky, Pyotr Ilyich	1,5	0,3	0,1	5,6	0,6	0,8	5,4	3,2	5,3	5,0	4,3	9,4	7,2	9,2	5,0	30,3	6,9
1841	Dvorak, Antonin	0,2	0,3	0,2	3,6	0,7	0,5	3,1	2,4	3,7	4,4	2,7	10,6	5,5	3,9	7,1	8,0	43,2
		Byrd, William	Victoria, Tomas Luis de	Lully, JeanBaptiste	Scarlatti, Alessandro	Couperin, Francois Le Grand	Dandrieu, JeanFrancois	Bach, Johann Sebastian	Handel, Friedrich Georg	Haydn, Franz Joseph	Mozart, Wolfgang Amadeus	Beethoven, Ludwig van	Schubert, Franz Peter	Liszt, Franz	Alkan, Charles Henri Valentin	Brahms, Johannes	Tchaikovsky, Pyotr Ilyich	Dvorak, Antonin
		Renaissance		**Baroque**						**Classicism**			**Romanticism**					

Table 4.8: Confusion matrix by composers using a works threshold of 100 and Random Forest classifier. The values are presented in %. The green highlighted values mean a better class classification; the red highlighted values mean a worse classification.

1543	Byrd, William	81,2	8,0	2,9	0,3	2,2	1,6	1,5	0,8	0,4	0,3	0,5	0,0	0,2	0,0	0,1	0,1	0,1
1548	Victoria, Tomas Luis de	8,4	81,3	3,3	0,2	0,6	0,9	0,8	0,8	0,6	0,2	0,7	0,2	0,6	0,2	0,6	0,3	0,3
1632	Lully, JeanBaptiste	3,5	3,0	65,8	0,9	5,5	3,2	3,2	4,2	1,4	1,9	1,8	1,5	0,9	0,9	1,7	0,4	0,1
1660	Scarlatti, Alessandro	0,9	0,6	1,2	59,3	3,4	4,7	2,6	2,1	3,7	5,4	3,0	2,2	0,7	1,5	2,8	3,4	2,5
1668	Couperin, Francois Le Grand	1,4	0,5	4,7	4,8	59,4	12,0	4,1	4,3	1,2	1,4	0,9	0,0	0,3	0,3	2,5	1,2	1,2
1682	Dandrieu, JeanFrancois	3,0	2,4	5,0	5,2	11,7	50,2	5,5	3,8	2,6	2,1	2,7	0,6	0,6	1,8	1,2	1,4	0,4
1685	Bach, Johann Sebastian	3,5	1,9	3,6	6,3	8,7	5,0	45,1	6,1	1,7	3,1	2,4	1,2	0,7	3,0	3,5	2,6	1,7
1685	Handel, Friedrich Georg	2,6	2,8	10,5	6,1	7,4	9,0	11,1	21,6	5,1	4,1	3,6	2,9	2,0	2,2	3,9	2,0	2,9
1732	Haydn, Franz Joseph	1,3	0,2	2,0	4,1	1,7	5,7	2,9	4,8	32,6	18,1	6,4	4,3	4,0	2,1	4,6	3,8	1,5
1756	Mozart, Wolfgang Amadeus	1,4	1,4	2,8	5,2	2,5	1,8	3,7	3,3	18,3	32,5	9,0	3,7	0,6	2,3	5,3	3,1	3,3
1770	Beethoven, Ludwig van	0,9	2,9	6,1	6,1	2,0	2,6	2,2	1,7	8,8	11,7	24,3	10,6	5,0	3,6	4,0	2,9	4,4
1797	Schubert, Franz Peter	0,1	0,7	0,6	2,8	0,0	0,6	1,4	1,1	3,9	3,9	7,4	37,4	6,3	7,0	10,9	6,6	9,3
1811	Liszt, Franz	0,2	0,1	1,9	1,9	0,4	1,7	0,9	1,1	2,0	1,9	4,2	6,0	55,2	8,2	4,1	4,4	5,7
1813	Alkan, Charles Henri Valentin	1,6	0,7	0,8	2,8	0,9	1,2	4,2	1,2	2,1	2,8	5,5	8,1	15,4	27,0	8,0	11,4	6,2
1833	Brahms, Johannes	0,2	0,0	1,7	2,1	1,6	2,6	4,0	2,8	3,5	1,8	1,3	9,0	7,5	4,0	44,1	3,9	10,0
1840	Tchaikovsky, Pyotr Ilyich	1,0	1,0	1,4	5,1	1,4	1,0	3,4	2,7	3,2	4,2	2,9	7,2	5,1	9,2	4,5	34,8	11,9
1841	Dvorak, Antonin	0,3	0,4	0,3	4,7	0,6	0,3	1,8	2,0	2,0	1,8	2,0	6,1	7,3	6,1	14,2	7,8	42,2
		Byrd, William	Victoria, Tomas Luis de	Lully, JeanBaptiste	Scarlatti, Alessandro	Couperin, Francois Le Grand	Dandrieu, JeanFrancois	Bach, Johann Sebastian	Handel, Friedrich Georg	Haydn, Franz Joseph	Mozart, Wolfgang Amadeus	Beethoven, Ludwig van	Schubert, Franz Peter	Liszt, Franz	Alkan, Charles Henri Valentin	Brahms, Johannes	Tchaikovsky, Pyotr Ilyich	Dvorak, Antonin
		Renaissance		**Baroque**						**Classicism**			**Romanticism**					

Table 4.9: Confusion matrix by composers using a works threshold of 100 and SVM classifier. The values are presented in %. The green highlighted values mean a better class classification; the red highlighted values mean a worse classification.

Renaissance	64,9	31,7	1,3	1,6	0,5
Baroque	13,5	72,3	5,0	6,7	2,4
Classicism	5,1	25,7	42,0	23,8	3,5
Romanticism	2,2	17,7	10,9	59,2	10,0
20th Century	1,6	18,5	6,4	31,5	42,1
	Renaissance	**Baroque**	**Classicism**	**Romanticism**	**20th Century**

Table 4.10: Confusion matrix by music periods using a works threshold of 20 and Random Forest classifier. The values are presented in %. The green highlighted values mean a better class classification; the red highlighted values mean a worse classification.

Renaissance	50,1	41,3	3,3	3,6	1,7
Baroque	11,8	71,5	5,3	8,4	3,1
Classicism	3,8	31,1	34,9	24,3	5,9
Romanticism	2,6	16,9	7,2	59,9	13,3
20th Century	2,8	16,4	5,1	32,5	43,3
	Renaissance	**Baroque**	**Classicism**	**Romanticism**	**20th Century**

Table 4.11: Confusion matrix by music periods using a works threshold of 20 and SVM classifier. The values are presented in %. The green highlighted values mean a better class classification; the red highlighted values mean a worse classification.

4.3 Scalability analyses

In this section we will perform analysis in order to find a relation the effect on accuracy by the number of composers, on one side, and the number of included works on the other side.

4.3.1 Increasing the number of composers

In this section we perform a set of experiments in order to analyse how the system reacts when we modify the number of composers, but we keep all the other parameters, including the number of used works. In this experiments we will use 20 works in all the classifications in order to use the wider number of composers.

In Figure 4.3 we present the graph of the results, including the 3 classifications. In all the classifications, Random Forest achieves the best results in all the composers range. It creates a perfect smooth curve, which achieves almost a 90% in the binary classifications (2 composers). On the other hand, SVM classification have some slightly fluctuations between 20 and 60 composers. From this table we also extract that we achieve the similar accuracy results for the classification of 10 composers (around 60%) where other studies from the state of the art classify 5 composers. In addition, our accuracies in the classification of 5 composers (around 70%) is higher these studies as well.

In the classification of more than 100 composers, we observe the asymptotic tendency of the accuracy, which is around 30% for Random Forest and 25% for SVM. On the other hand, if we use more than 20 composers in this task, the classification is more complex and the results gets worse (less than 45% of accuracy); because of that, in the studies related to big collections, the minimum of composers should be 20 in order to improve these cases.

4.3.2 Increasing the number of pieces per composer

In this section we will perform another analysis, this time related to the variation of number of works per composer used in the classification, fixing the number of composers.

In these experiments we use the composers group that have a minimum number of 100 works, in order to be able to perform the wider classification set. However, we need to use different configurations in our experiments, depending the number of used works, using the same number of works in the test set in order to avoid biased results. The configuration parameters are presented in Table 4.3.2. The number of works per training and test set are calculated depending of the K number for the K-Fold cross validation and the total number of used works in the

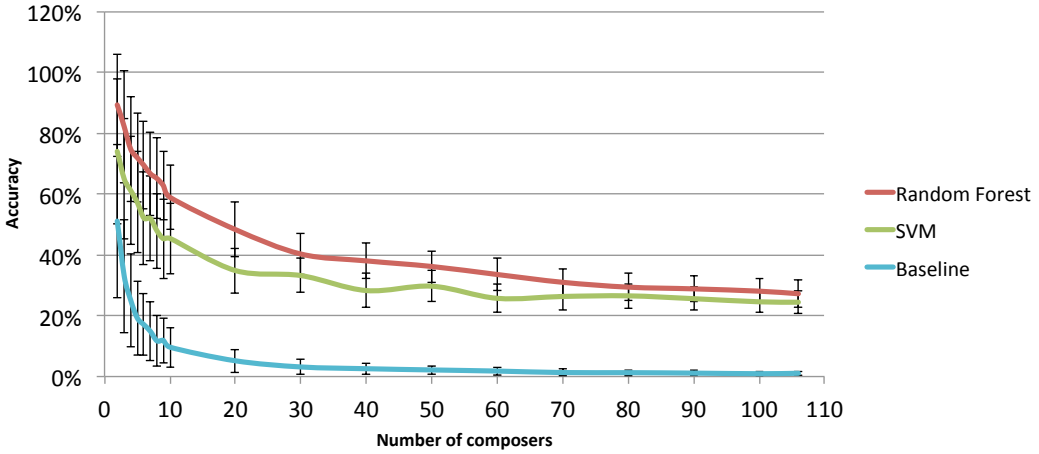

Figure 4.3: Principal group features combinations using Random Forest Classifier. For each classification there is the combination identification (using the numbers corresponding to the ranking position in Table 4.5) and its classification accuracy.

experiment. The smaller is the difference between works number of training and test sets, the higher are the number of runs per classification.

Number of works				
Total	**Training Set**	**Test set**	**K-Folds [K number]**	**N. Runs**
100	90	10	10	10
90	80	10	9	20
80	70	10	8	30
70	60	10	7	40
60	50	10	6	50
50	40	10	5	60
40	30	10	4	70
30	20	10	3	80
20	10	10	2	90

Table 4.12: Classifications setup for scalability analysis of works

In Figure 4.4 we present the graph of the achieved results, including all the classifications used in this study. We want to remind that the number of composers in this experiment is 17. In the analysis of the results, when we use a lower number of works for the classification, the best classifier is Random Forest, whereas when we use a highest number of works, the best results achieved are performed by SVM. We find the lines crossing in the classification of 60 works.

The classification using SVM seems to keep being improved if we see the

direction of the trend line, whereas for the Random Forest classifier, the trend line starts to be flatter in the highest part of the graph. On the other hand, the base lines remains fat during all the experiment because it is based only to the random classification of all the instances (works) over 17 classes (composers).

From this experiment, we see that the larger is the number of works used for the classification, the higher are the accuracies, However, we would need to perform more experiments increasing the number of used works, in order to confirm the ceiling effect that insinuates in the classification with the usage of 100 works. In the case of this ceiling effect appears, this means that we achieved the best possible classification with these features, thus, we would need to create new features or redefine the used features (or their parameters).

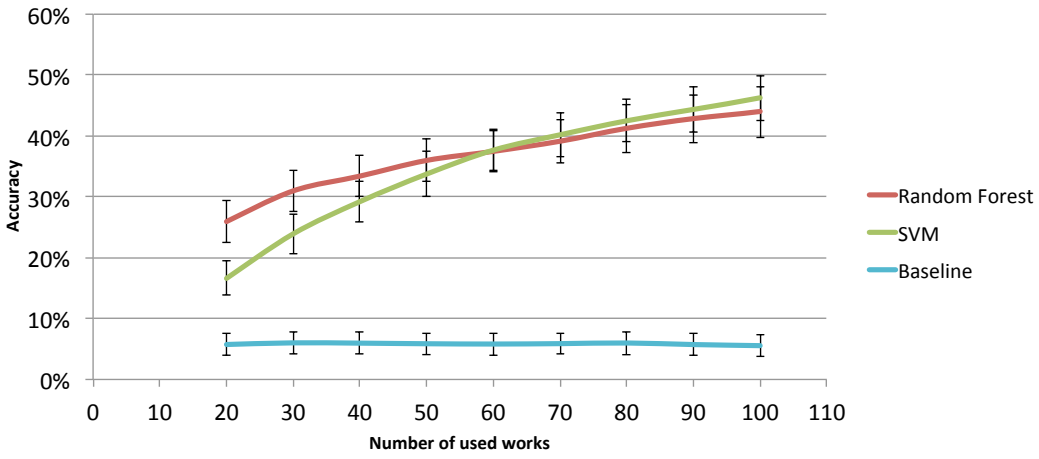

Figure 4.4: Principal group features combinations using Random Forest Classifier. For each classification there is the combination identification (using the numbers corresponding to the ranking position in Table 4.5) and its classification accuracy.

Chapter 5

CONCLUSIONS AND FUTURE WORK

From this study we would want to extract some conclusions from our results and propose some future work that could be done in order to achieve better results or extend it.

First of all we want to say that one of the most important parts of studies like this is to have a database properly filtered and documented: we developed a lot of time in this step because bad information about the works or works with errors may have an impact in our results. We also take into account that some of the estimated keys are not correct; this could also had an impact to the classifications experiments as well.

Taking an analysis on the results of the separated feature groups accuracies comparing to the ones after the feature selection, we noticed that the results are quite similar; in the best of the improvements, this percentage is 16,8%. If we take into account absolute values, this percentage doesn't seem to be relevant, but if we make a relative comparison, this improvement is 56% higher than the accuracy without feature selection.

From the confusion matrix we can extract some useful information: as we have seen in the tables, the selected features gives prominent accuracies in the classification of the two Renaissance composers that we have in the works threshold of 100 composers: *W. Bird* and *T.L. de Victoria*, achieving more than 80% of accuracy for both composers. One the other side, composers like *L. van Beethoven* (Classicism) or C.H.V. Alkan (Romanticism) have accuracies lower than 30%. The confusion matrix of the music periods also give us an overall vision of utility per each music period from the features combination. However, we have to take into account that the number of composers per period is not the same, so the results could be biased.

The scalability analyses give us information about how the system works in

several conditions. From the analysis of the variation of number of works we could say that if we have less than 60 works, we should choose the Random Forest classifier, which performs a better information generalization, whereas in the case of more than 60 works, SVM performs a better classification, which is more appropriate for larger data experiments.

The accuracies achieved in this study, improve most of the results that we have presented in the related work, even taking into account the generalization for a large number of composers. However, we would like to propose some future work that we think that could be interesting in order to improve the results and perform and extension of the study carried out.

The Chromagram is the starting point for all the extracted features. In this study, the temporal unit for the Chromagram construction is 1 beat per column. The variation of this parameter would lead to changes in the extracted features and in the classification accuracies as well, because the information of the Chromagram would be increased or decreasing due to the splitting or merging of columns of it.

In some feature we set up the Chromagram threshold parameter. This parameter selects which notes are used, in terms of their importance during the beat, by the features. In most of the cases, we only have used 0, 0.5 or 1 for the feature extraction. Other values could be used, in order to check how the classifications reacts to them.

Most of the features are related to the harmony or the chroma class distribution over the piece. We could create feature groups related to the rhythm, although these ones might need another first approach, different to the Chromagram, or a new Chromagram with a reference duration much shorter than a beat, e.g. a 16th note or semiquaver. Then, we should be able to look for repetition patterns of chroma classes depending on a more exact duration and location over the bars.

In the feature selection, we only have selected combinations of feature groups. However, these feature groups have single features which are used in the classifications. Using the weights that the classifier extract with each combination, we could try to discover which are the explicit features from each feature group that are more useful in the classifications.

In the analysis of the results, we concluded that the designed features have their weakest point in the classification of composers from the classical period. Extended experiments using only composers from this music period would help us to find why the system doesn't work properly for these composers, and try to design another feature which helps to achieve better results.

Appendix A

COMPOSERS LIST

In this appendix we present the complete list of the composers name with their respective number of works that we used from the filtered data set. The list is sorted in descendent order by works number per composer.

Table A.1: Complete composers list

Composer name	Number of works
Bach, Johann Sebastian	1368
Haydn, Franz Joseph	568
Scarlatti, Alessandro	555
Mozart, Wolfgang Amadeus	494
Handel, Friedrich Georg	480
Beethoven, Ludwig van	463
Victoria, Tomas Luis de	333
Schubert, Franz Peter	246
Tchaikovsky, Pyotr Ilyich	238
Alkan, Charles Henri Valentin	237
Dandrieu, Jean-Francois	205
Dvorak, Antonin	142
Brahms, Johannes	137
Couperin, Francois Le Grand	115
Lully, Jean-Baptiste	112
Liszt, Franz	109
Byrd, William	109
Debussy, Claude Achille	97
Schumann, Robert Alexander	97
Brade, William	93

Schutz, Heinrich	92
Morales, Cristobal de	88
Chopin, Fryderyk Francisek	86
Guerrero, Francisco	84
Buxtehude, Dietrich	83
Faure, Gabriel	83
Schein, Johann Hermann	83
SaintSaens, Camille	82
Resenmuller, Johann	79
Shostakovich, Dimitri	79
Scriabin, Alexander Nikolayevich	78
Palestrina, Giovanni Pierluigi da	68
Anglebert, JeanHenri	67
Holborne, Anthony	65
Soler, Antonio	65
Lasso, Orlando di	64
Markull, Friedrich Wilhelm	62
Albeniz, Isaac Manuel Francisco	61
Dowland, John	61
Frescobaldi, Gerolamo	60
Raff, Johann Joachim	60
Cima, Giovanni Paolo	56
Franck, Cesar-Auguste	56
Corelli, Arcangelo	53
Pachelbel, Johann Christoph	53
Telemann, Georg Philipp	53
Maier, Michael	50
Albinoni, Tomaso Giovanni	49
Boyvin, Jacques	47
Chaumont, Lambert	47
Raehs, Morten	46
Mendelssohn Bartholdy, Felix	44
Ravenscroft, Thomas	44
Hammerschmidt, Andreas	42
Scarlatti, Domenico	42
Clementi, Muzio	41
Eccard, Johannes	41
Krebs, Johann Ludwig	40

Uccellini, Marco	40
Vivaldi, Antonio Lucio	40
Gesualdo da Venosa, Carlo	37
Busoni, Ferruccio Dante Michelangiolo Benvenuto	36
Cocquiel, J.I.J.	36
Danzi, Franz	36
Gottschalk, Louis Moreau	36
Becker, Dietrich	34
Fischer, Johann Kaspar Ferdinand	34
Froberger, Johann Jakob	32
Dornel, LouisAntoine	31
Merula, Tarquinio	31
Simpson, Thomas	31
Bruckner, Joseph Anton	30
Groh, Johann	30
LefebvreWely, Louis James Alfred	30
Pederson, Mogens	30
Young, William	30
CastelnuovoTedesco, Mario	29
Reger, Max	29
Stravinsky, Igor Fyodorovich	29
Vecchi, Orazio	29
Clerambault Louis-Nicolaus	28
Mahler, Gustav	28
Lemmens, Jacques-Nicolas	27
Delibes, Clement Philibert Leo	26
Rameau, Jean-Philippe	26
Corrette, Gaspard	25
Couperin, ArmandLouis	25
Hummel, Johann Nepomuk	25
Jacquet de la Guerre, Elisabeth	25
Fullsack Hildebrandt, publishers	24
Paganini, Niccolo	24
Pozzoli, Ettore	24
Reicha, Anton	24
Janacek, Leos	22
Lecuona y Casado, Ernesto	22
Adson, John	21

Bach, Johann Christian	21
Bizet, Georges Alexandre Cesar Leopold	21
Brachrogge, Hans	21
Mainerio, Giorgio	21
Offenbach, Jacques	21
Praetorius, Michael	21
Roman, Johan Helmich	21
Satie, Alfred Erik Leslie	21
Altenburg, Michael	20
Monteverdi, Claudio	20

Appendix B

CHORD DICTIONARY

In Figure B.1 there is the chord dictionary used in this study with the name of each chord.

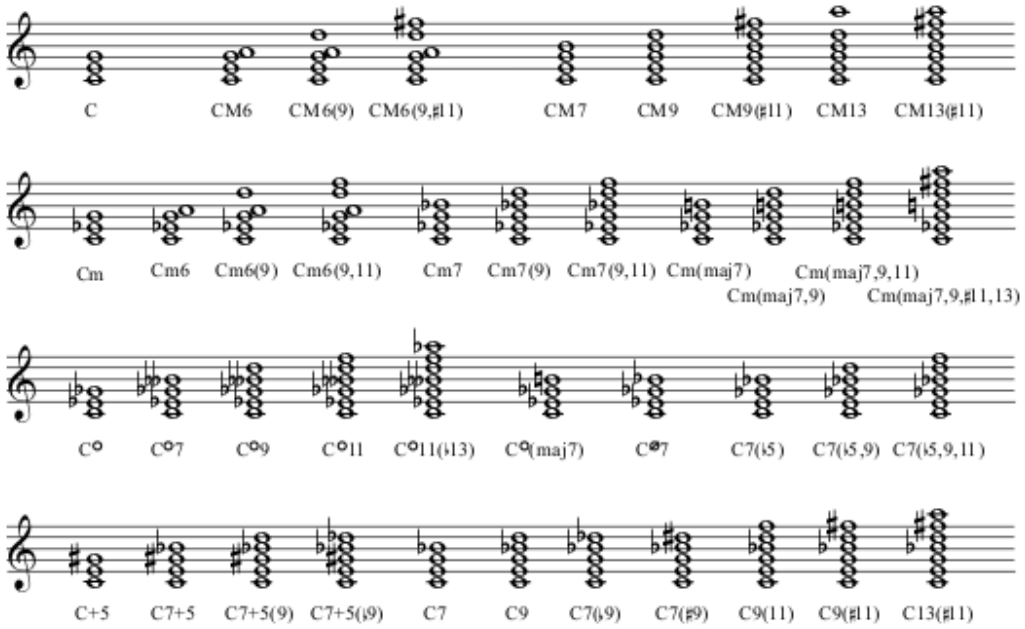

Figure B.1: Chord dictionary with all the combinations used in this study of 3 or more notes, with root C.

In Table B.1 there are the complete list of the chord profile vectors used in this study. Each on of the numbers of the first column represents presence (1) or missing (0) of the chroma class in the chord. In the third column there is the correspondence of the chord with its simplified version.

Table B.1: Chord dictionary profiles

Binary chord profile	Chord type	Simplified chord type
1 0 0 0 0 0 0 0 0 0 0 0	Perfect unison	Perfect unison
1 1 0 0 0 0 0 0 0 0 0 0	Minor 2nd or Major 7th	Minor 2nd or Major 7th
1 0 1 0 0 0 0 0 0 0 0 0	Major 2nd or Minor 7th	Major 2nd or Minor 7th
1 0 0 1 0 0 0 0 0 0 0 0	Minor 3rd or Major 6th	Minor 3rd or Major 6th
1 0 0 0 1 0 0 0 0 0 0 0	Major 3rd or Minor 6th	Major 3rd or Minor 6th
1 0 0 0 0 1 0 0 0 0 0 0	Perfect 4th or Perfect 5th	Perfect 4th or Perfect 5th
1 0 0 0 0 0 1 0 0 0 0 0	Tritone	Tritone
1 0 0 0 1 0 0 1 0 0 0 0	M	M
1 0 0 1 0 0 0 1 0 0 0 0	m	m
1 0 0 1 0 0 1 0 0 0 0 0	dim	dim
1 0 0 0 1 0 0 0 1 0 0 0	aug	aug
1 0 0 0 0 1 0 1 0 0 0 0	sus	sus
1 0 0 0 1 0 0 1 0 0 1 0	7	M
1 0 0 1 0 0 0 1 0 0 1 0	m7 or M6	m
1 0 0 0 1 0 0 1 0 0 0 1	M7	M
1 0 0 1 0 0 0 1 0 0 0 1	m(maj7)	m
1 0 0 1 0 0 0 1 0 1 0 0	m6 or dim(min7)	m
1 0 0 0 1 0 0 0 1 0 1 0	aug7	aug
1 0 0 1 0 0 1 0 0 1 0 0	dim7	dim
1 0 0 1 0 0 1 0 0 0 0 1	dim(maj7)	dim
1 0 0 0 1 0 1 0 0 0 1 0	7(b5)	NC
1 0 0 0 0 1 0 1 0 0 1 0	7sus4	sus
1 0 0 0 0 1 0 1 0 0 0 1	M7sus4	sus
1 0 1 0 0 0 0 1 0 0 1 0	7sus2	sus
1 0 1 0 0 0 0 1 0 0 0 1	M7sus2	sus
1 0 1 0 1 0 0 1 0 0 0 0	add2	M
1 0 1 1 0 0 0 1 0 0 0 0	madd2	m
1 0 1 0 1 0 0 1 0 0 1 0	9	M
1 0 1 1 0 0 0 1 0 0 1 0	m7(9)	m
1 1 0 0 1 0 0 1 0 0 1 0	7(b9) or dim9	M
1 0 0 1 1 0 0 1 0 0 1 0	7(#9)	NC
1 0 1 0 1 0 0 1 0 0 0 1	M9	M
1 0 1 1 0 0 0 1 0 0 0 1	m(maj7 9)	m
1 0 1 0 1 0 0 1 0 1 0 0	M6(9)	M
1 0 1 1 0 0 0 1 0 1 0 0	m6(9)	m
1 0 1 0 1 0 1 0 0 0 1 0	7(b5 9) or aug7(9)	NC

1 1 0 0 1 0 0 0 1 0 1 0	aug7(b9)	aug
1 0 1 0 1 1 1 0 0 0 1 0	7(b5 9 11)	NC
1 0 1 0 1 1 0 1 0 0 1 0	9(11) or M6(9 #11)	NC
1 0 1 0 1 0 1 1 0 0 1 0	9(#11)	NC
1 0 1 1 0 1 0 1 0 0 1 0	m7(9 11) or M13	m
1 0 1 1 0 1 0 1 0 0 0 1	m(maj7 9 11)	m
1 0 1 1 0 1 1 0 0 1 0 0	dim11	dim
1 0 1 0 1 0 1 1 0 0 0 1	M9(#11)	M
1 0 1 1 0 1 0 1 0 1 0 0	m6(9 11)	m
1 0 1 0 1 0 1 1 0 1 1 0	13(#11)	NC
1 0 1 0 1 0 1 1 0 1 0 1	M13(#11)	NC
1 0 1 1 0 0 1 1 0 1 0 1	m(maj7 9 #11 13)	NC
1 0 1 1 0 1 1 0 1 1 0 0	dim11(13)	NC

References

[1] H. a. Harding, *Analysis of Form in Beethoven's Sonatas*. 1901.

[2] P. Melidis, *Electronic Music Artist Identification*. PhD thesis, 2012.

[3] O. Lartillot and P. Toivianen, "A Matlab Toolbox for Musical Feature Extraction from Audio," in *International Conference on Music Information Retrieval*, 2007.

[4] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, and O. Mayor, "Essentia: an Audio Analysis Library for Music Information Retrieval," in *Internacional Society for Music Information Retrieval Conference*, pp. 493 – 498, 2013.

[5] M. Haro, J. Serrà, P. Herrera, and A. Corral, "Zipf's Law in Short-Time Timbral Codings of Speech, Music, and Environmental Sound Signals," *PLoS ONE*, vol. 7, no. 03/2012, p. e339933, 2012.

[6] S. R. Baek, "Music Genre / Mood / Composer classification : MIREX 2014 Submissions," *Mirex2014, extended abstract*, 2014.

[7] S.-C. Lim, B. Karam, L. Jong-Soel, J. Sei-Jin, and K. Moo Young, "Pooled features classification," *Submission to music genre/mood classification for MIREX 2012*, 2012.

[8] J. Lebar, G. Chang, and D. Yu, "Classifying Musical Scores by Composer:," in *Proceedings of 3rd international workshop on Machine learning and music - MML '10*, pp. 37–40, 2010.

[9] L. Mearns, D. Tidhar, and S. Dixon, "Characterisation of composer style using high-level musical features," in *Proceedings of 3rd international workshop on Machine learning and music - MML '10*, vol. 45, p. 37, ACM Press, 2010.

[10] E. Backer and P. V. Kranenburg, "On musical stylometry-a pattern recognition approach," in *Pattern Recognition Letters*, vol. 26, pp. 299–309, Feb. 2005.

[11] L. Peusner, "A Graph Topological Scores Representation of Melody Scores," *Leonardo Music Journal*, vol. 12, pp. 33–40, 2002.

[12] M. Li and R. Sharp, "Melody classification using a similarity metric based on kolmogorov complexity," *Conference on Sound and Music Computing*, 2004.

[13] R. Cilibrasi and P. M. B. Vitányi, "Clustering by compression," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

[14] E. Pollastri, V. Comelico, and G. Simoncelli, "Classification of Melodies by Composer with Hidden Markov Models," *Distribution*, 2001.

[15] R. Hillewaere, B. Manderick, and D. Conklin, "Melodic models for polyphonic music classification," *Proceedings of the 2nd International Workshop on Machine Learning and Music (MML 2009)*, pp. 31–36, 2009.

[16] P. Ponce de León and J. Iñesta, "Statistical description models for melody analysis and characterization," in *International Computer Music Conference*, pp. 149 – 156, 2004.

[17] B. Jesser, *Interaktive Melodieanalyse*. 1991.

[18] C. McKay and I. Fujinaga, "Automatic genre classification using large high-level musical feature sets," in *International Conference on Music Information Retrieval*, pp. 525–530, 2004.

[19] M. a. Kaliakatsos-Papakostas, M. G. Epitropakis, and M. N. Vrahatis, "Weighted Markov chain model for musical composer identification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6625 LNCS, no. PART 2, pp. 334–343, 2011.

[20] T. Hedges, P. Roy, and F. Pachet, "Predicting the Composer and Style of Jazz Chord Progressions," *Journal of New Music Research*, vol. 43, no. 3, pp. 276–290, 2014.

[21] Y.-W. Liu and E. Selfridge-field, "Modeling Music as Markov Chains : Composer Identification," tech. rep., Standford, 2006.

[22] M. a. Kaliakatsos-Papakostas, M. G. Epitropakis, and M. N. Vrahatis, "Musical composer identification through probabilistic and feedforward neural networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6025 LNCS, pp. 411–420, 2010.

[23] T. Pérez García, P. J. Ponce de León, and M. Sanchez Molina, "Standard MIDI File to Text," in *Pattern Recognition and Artificial Intelligence Group. University of Alicante.*, 2011.

[24] D. Temperley, "What's Key for Key? The Krumhansl-Schmuckler Algorithm Key-Finding Algorithm Reconsidered," *Music Perception*, vol. 17, no. 1, pp. 65–100, 1999.

[25] D. Boswell, "Introduction to Support Vector Machines," 2002.