Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

GenFastLAS: Scaling up the FastLAS system to large feature spaces, and application to Symbolic Music Genre Classification

Author:
Lancelot Blanchard

Supervisor:
Prof. Alessandra Russo

Second Marker:
Prof. Francesca Toni

June 20, 2022

Abstract

As humans increasingly turn towards Artificial Intelligence to assist them in a variety of tasks, machines are required to be able to interact with their users in a straightforward and intelligible way. This is especially true in areas relating to human creativity, research, or education. Additionally, following the expanding availability of data, Machine Learning systems need to be able to handle complex and multifaceted forms of data in an accurate and efficient manner. On the one hand, while Logic-based Learning has been proposed as a way to bring explainability to Machine Learning, it often suffers from a lack of scalability with respect to the complexity of the data it is provided with. On the other hand, Genetic Algorithms have been developed in order to learn from complex data by efficiently exploring the hypothesis space. We, therefore, develop GenFastLAS, a Logic-based Learning system, which can learn explainable solutions from complex data by using Genetic Algorithms to explore the symbolic hypothesis space. We then use our system to explore the task of Symbolic Music Genre Classification in a newly explainable, interactive, and efficient way. To do so, we introduce a set of novel features for symbolic music data, together with a systematic way to compute them. Through these contributions, we hope to pave the way for a growth in Machine Learning applications of symbolic data, particularly symbolic music, which can undoubtedly bring value to the areas of musicology research and education, as well as enhance human creativity.

Acknowledgements

I would first like to thank my supervisor, Prof. Alessandra Russo, for her invaluable help and constant availability throughout this project. I would also like to deeply thank everyone who kindly answered the survey I conducted for the evaluation of this project. Finally, I would like to express my most sincere gratitude towards my family, my friends, and everyone who selflessly took the time to support me, directly or indirectly, over the course of this project, and of my entire degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

1.1.1 Developing a scalable and explainable Machine Learning system

Over the past three decades, many Machine Learning (ML) methods have emerged as valuable techniques to solve a multitude of problems. ML has been successfully applied to a wide variety of tasks, with aims as diverse as problem solving, industrial optimisation, or scientific modelling, and the continuous development of new technologies keeps on improving the accuracy and speed of those tasks. However, research in Machine Learning has been consistently confronted to two significant challenges:

(1) While statistical ML has been able to wonderfully solve many complex problems, the solution they offer us is often obscure and non-explainable. This is a considerable drawback, especially in domains where explainability is a core requirement of the problem solving process, such as research, healthcare, justice, creative work, or education. Explainability becomes even more desirable when attempting to create a stronger interactivity between humans and AI systems, since it then becomes essential for both entities to be able to communicate. We would therefore like ML systems to learn solutions and knowledge that can be communicated and transferred to human users in a straightforward way.

(2) Additionally, real-world data are often complex and multifarious, and it is rarely straightforward, even for humans, to be able to reason with a limited set of features. The recent advent of web services and online telemetry have resulted in the availability of extensive amounts of data, and have opened up a world of application opportunities for ML. However, extracting meaningful information out of this overwhelming amount of data can sometimes prove to be very challenging, and systems need a way to explore this large space efficiently. As such, we would like to be able to consider data in their entirety, with an access to as many features as possible. A problem that naturally arises is that it is often very difficult for ML technologies to perform well with too many features to consider. Ideally, we would like our ML systems to intelligently select relevant features.

Multiple systems have been developed to approach those two distinct desiderata, and several successful solutions have emerged. For example, Symbolic Artificial Intelligence and Logic-based Learning technologies have been suggested as a way to perform Machine Learning using a humanly understandable reasoning system, such as First-Order Logic, tackling desideratum (1). In another context of research, Genetic Algorithms, and other classes of Evolutionary Algorithms, have successfully been used to explore large feature spaces in an efficient way by taking inspiration from the behaviour of living organisms, tackling desideratum (2). In particular, Quality-Diversity (QD) algorithms are able to perform this exploration by focusing not only on optimality, but also on diversity, which is especially valuable in the areas that we are interested in, such as research or education. However, there is no current solution, to the best of our knowledge, that have been proposed to efficiently combine both of these types of technology in order to solve complex problems with large feature spaces in an explainable way. This naturally leads to a first Research Question:

Research Question 1. *Can we develop a Machine Learning system that can efficiently be applied to complex and multifaceted data, and learn explainable solutions?*

1.1.2 Enabling explainable uses of symbolic music data through new features

As previously mentioned, explainable and interactive applications of Machine Learning technologies offer promising opportunities for a variety of purposes, such as research, creative work, or education. In the domain of music, Machine Learning has been largely applied to raw audio data for industrial purposes, especially, and most recently, within music streaming and radio companies. However, music can also be frequently found in its symbolic form, as written sheet, tablatures, or in MIDI files (see Section 2.4). This type of data, although less obviously applicable to industry aims, embeds a lot of musical insight and music theory knowledge due to its symbolic format. As such, symbolic music can be of great value for the purposes of research and education in musicology, or for the enhancement of human creativity.

When observing the progress of Music Information Retrieval (MIR) research in recent year, it appears that symbolic music data have not been used nearly as extensively as raw audio data in Machine Learning applications. Yet, it is evident that symbolic music data, through their abundance and their ability to be understood by human users, offer a valuable opportunity to perform typical MIR tasks in an explainable and interactive way. Such tasks can include the composition of musical pieces in collaboration with human users, the detailed construction of user profiles used for music recommendation, or the classification of music based on music theory knowledge.

Given their appeal, the lack of such systems is most probably explained by the prior lack of a framework that enables such explainable learning in an efficient way, as previously stated. Symbolic music data are indeed complex, and many features have been developed for the application of various MIR tasks. However, these features are often very specific and in a numerical format, which impairs their usefulness as explainable features. Under the assumption that an efficient and explainable learning framework can exist, it is now required for symbolic music to be accompanied with naturally explainable features, that reflect the way humans tend to reason about music and music theory. As such, this observation gives birth to a second Research Question:

Research Question 2. *Can we design a set of explainable features for symbolic music, that can follow the manner in which humans reason about music, and enable explainable and interactive applications of MIR tasks?*

1.2 Objectives

Given the previously stated research questions, it is clear that this project gravitates around two main objectives, which both combine into a final third one:

1. Firstly, we aim to develop a Logic-based Learning system that can process symbolic data and learn explainable models, in a way that is efficient, and that can easily scale up with respect to the complexity of the data.

2. Secondly, we wish to design novel features that can easily be used for the processing of symbolic music data, and that can embed explainability by echoing music theory background knowledge. Ideally, these features should provide a common ground for communication between humans and machines, and, thus, enable a stronger interactivity between both entities.

3. Thirdly, and finally, we aspire to combine our new learning system together with our novel features in order to perform Symbolic Music Genre Classification in an innovative way. Our newly learnt model, therefore, aims to provide us with better insight into the theoretical musical mechanisms behind the task of genre classification, as well as pave the way for future interactive applications.

1.3 Contributions

In Chapter 3, we introduce *GenFastLAS*, a scalable Logic-based Learning system which makes use of the MAP-Elites QD algorithm [1] in order to cleverly guide the feature space exploration of the FastLAS learning system [3]. Throughout this chapter, we introduce a novel and generalisable genetic encoding applicable to any Learning from Answer Sets task. We also present a modified version of the MAP-Elites algorithm which enables the processing of discrete genotypes and facilitates our integration with FastLAS. We finally introduce a couple of behaviour descriptors that articulate well with FastLAS tasks, and discuss a new research area for the automatic generation of symbolic behaviour descriptors that can enable better interactive usages of explainable AI learning systems. In Chapter 4, we introduce a set of eight novel features for the use of symbolic music data in explainable AI tasks, together with a systematic approach for their computation. We provide a new discussion of the specifications and requirements behind the design of such explainable symbolic music features, and touch upon the potential of those new features in the areas of musicology research and education.

Chapter 2

Background

In this chapter, we introduce some background for the key concepts that this thesis considers.

2.1 Explainable Artificial Intelligence

Since its first theoretical conception, Artificial Intelligence (AI) has been formulated as an ensemble of methods that all aim to help humans complete tasks and solve problems. Recently, Machine Learning systems have proven to be successful in achieving many of those aims. However, several of those aims are still unachievable by such systems due to the complexity and lack of transparency of their output. In fact, many areas, like healthcare, justice, research, education, or creativity, require a high degree of trust, and AI systems can only be used if they feature transparency and provide a common ground to enable communication with human users [4]. As such, building explainable AI systems has been an important challenge within the area of Artificial Intelligence research.

Miller defines an AI system to be explainable if it takes into account, during the computation, how well humans will be able to understand its output [5]. Explainability is then expressed as the ability for a system to give explanations, which is defined by two processes: a cognitive process, which aims to determine the reasoning behind a particular task or phenomenon, and a social process, which aims to transfer the knowledge of this reasoning to another entity. Finally, in order for this transfer to be correctly performed, there needs to be a common ground for communication which enables both entities to exchange information easily [6]. This further motivates not only the need to have explainable AI systems, but additionally to make use of data that embed some dimension of explainability within them.

2.2 Learning from Answer Sets

Taking its roots at the intersection of Knowledge Representation and Machine Learning, Logic-based Learning has been suggested as a method to perform Machine Learning tasks in a logically structured and explainable way. In particular, the Learning from Answer Set (LAS) framework [7] has been developed to perform this learning using programs expressed under the Answer Set Programming (ASP) semantics. In this thesis, we especially make use of the scalable version of this framework, called FastLAS [3]. We describe here some of the semantics and notations used within this framework.

We consider ASP programs which include three types of rules: normal rules, choice rules, and constraints. Given any atoms $\texttt{h}, \texttt{h}_1, \ldots, \texttt{h}_k, \texttt{b}_1, \ldots, \texttt{b}_n, \texttt{c}_1, \ldots, \texttt{c}_m$, we denote a rule of the form $\texttt{h :- b}_1\texttt{,} \ldots \texttt{,b}_n\texttt{,not c}_1\texttt{,} \ldots \texttt{,not c}_m$ as a *normal rule* R. $head(R) = \{\texttt{h}\}$ is the head of the rule, $body(R) = \{\texttt{b}_1\texttt{,} \ldots \texttt{,b}_n\texttt{,not c}_1\texttt{,} \ldots \texttt{,not c}_m\}$ is the body of the rule, and `not` here refers to negation as failure. An example of such a rule could be `sus_chord(C):-chord(C),not has_third(C)`, which states that any chord is suspended, unless it contains a third. The negated condition `not has_third(C)` is assumed to hold unless we can prove `has_third(C)` for some value of `C`. A *choice rule* is of the form $\texttt{l\{h}_1\texttt{,} \ldots \texttt{,h}_k\texttt{\}u :- b}_1\texttt{,} \ldots \texttt{b}_n\texttt{,not c}_1\texttt{,} \ldots \texttt{,not c}_m$ with `l` and `u` integers. We call $\texttt{l\{h}_1\texttt{,} \ldots \texttt{,h}_k\texttt{\}u}$ an *aggregate*. A *constraint* is of the form $\texttt{:- b}_1\texttt{,} \ldots \texttt{b}_n\texttt{,not c}_1\texttt{,} \ldots \texttt{,not c}_m$.

We also denote HB_P the *Herbrand Base* of an ASP program P. It is the set of all atoms that can be constructed using the predicate names, function symbols and constant symbols in P.

A subset of HB_P is called an (Herbrand) *interpretation* I of P if all elements of the subset are satisfied by I. Elements are satisfied as follows:

1. An atom $\texttt{a}$ is satisfied by an interpretation I if $\texttt{a} \in I$.
2. An aggregate $\texttt{l\{h}_1\texttt{,...,h}_k\texttt{\}u}$ is satisfied by an interpretation I if $\texttt{l} \leq |\texttt{\{h}_1\texttt{,...,h}_k\texttt{\}} \cap I| \leq \texttt{u}$.
3. For a rule R of the form $\texttt{h:-b}_1\texttt{,...,b}_n\texttt{,not c}_1\texttt{,...,not c}_m$, the body $body(R)$ is satisfied by an interpretation I if all positives atoms are satisfied by I (i.e. if $\forall i \in \{1, ..., n\}$, $\texttt{b}_\texttt{i}$ is satisfied by I) and none of the negated atoms are satisfied by I (i.e. if $\forall i \in \{1, ..., m\}$, $\texttt{c}_\texttt{i}$ is not satisfied by I.)

Given a program P with no aggregates in the heads of rules, we call a (Herbrand) *model* of P an interpretation I such that for every rule $R \in P$ such that the body $body(R)$ of the rule R is satisfied, the head $head(R)$ is also satisfied. Additionally, a model is called *minimal* if it has no strict subset which is also a model. We denote the *reduct* P^I of a program P with respect to an intepretation I the program constructed from the grounding of P in the following manner: first, we remove all the rules whose bodies contain a negated atom that is present in I; second, we remove all negative literals from the rules that remain; and we finally replace the head of any constraint rule with $\bot$. Finally, an answer set A of a program P is a minimal model of the reduct of P with respect to I. For programs with choice rules, we can apply the same semantics using a translation to normal rules described in Law et al. [8].

We finally define an *Observational Predicate Learning from Answer Sets* task as a tuple $T = \langle B, M, E^+ \rangle$, with B our background knowledge encoded as an ASP program, M our mode bias, and E^+ the set of our weighted context-dependent partial interpretations (WCDPIs). By definition, a WCDPI is a tuple $e_= \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$, with e_{id} an identifier for e, e_{pen} the penalty for not covering e that can either be a positive integer or ∞, e_{pi} the partial interpretation associated with e, and e_{ctx} the context of e expressed as an ASP program. The partial interpretation is further defined as $e_{pi} = \langle e^{inc}, e^{exc} \rangle$, with e^{inc} (rep. e^{exc}) the inclusions (resp. exclusions) of the partial interpretation e_{pi}. The mode bias is as well further defined as $M = \langle M_h, M_b \rangle$ where M_h (resp. M_b) are the sets of head (rep. body) mode declarations. We denote as N_h (resp. N_b) the size of the set M_h (resp. M_b). Finally, we define a scoring function $\mathcal{S}$. We say that an *hypothesis* $H \subseteq M$ is an *inductive solution* of T ($H \in ILP_{LAS}(T)$) if and only if we have: $\forall e \in E^+ \text{s.t.} e_{pen} = \infty$, $B \cup H$ accepts e.

2.3 Evolutionary and Genetic Algorithms

Evolutionary Algorithms (EAs) offer computational methods to solve optimisation problems by taking inspiration from the mechanisms of biological evolution, such as natural selection, mutation, and reproduction. A particular subset of those algorithms, called Genetic Algorithms (GAs), offers to do so by encoding the problem to solve into a biological simulation environment, with a precise set of semantics. In particular, the different candidate solutions are referred to as individuals inside a population, and are explored following specific rules. Each individual is characterised by its genotype, which represents the properties of the candidate solution. This genotype itself is made up of a set of genes, and is traditionally represented as a binary string. For each individual, we can then compute the value of the objective function in the optimisation problem, which represents the fitness of that individual. The evolution process is then simulated using different individuals that are initially generated, and a few genetic operations are carried out over the course of several generations. These operations include selecting individuals based on their fitnesses, simulating reproduction by combining the genotypes of two individuals, and emulating mutation by randomly perturbating the genotype of some individuals. Finally, the individuals with the best fitness are kept, and the corresponding solutions are returned.

Quality-Diversity (QD) algorithms make up another particular subclass of Evolutionary algorithms, and are focused on applying the same mechanisms as GAs by not only striving for the best fitness, but also by encouraging diversity in the set of returned solutions. By doing so, these algorithms are able to generate large collections of diverse solutions, which all perform the best within their own niche. Those niches are defined through specific behaviour descriptors that describe the solutions of individuals under the light of specific features of interest. This added aspect of diversity enables QD algorithms to be of particular use within areas that are concerned with more complex problems than optimisation alone, such as scientific research, or education.

2.4 Symbolic music and MIDI

While music as an art form is inherently embedded in an audio format, the exchange and communication of its content is often performed through symbolic representations. These can be sheet music, tablatures, or even lyrics, which all encode some part of a musical piece in a symbolic format. Within electronic systems, symbolic music is also widely used to enable creative applications, and, as such, many standards exist like MIDI, MusicXML, KERN, or GUIDO.

MIDI (Musical Instrument Digital Interface) [9] is the most common standard used to connect electronic musical instruments and computers, as well as to encode musical information in digital files. Used in MIR, it also allows us to have a symbolic representation of a piece of music in a compact form, and contains similar information as found on a musical score. When used in live performances, MIDI is used as a way to transfer the information of which and how notes are played to a virtual instrument, or synthesiser. Sound is then synthesised following the transmitted information. MIDI offers a variety of advantages over audio recordings, but also suffers from important weaknesses. As a matter of fact, MIDI cannot be used to perfectly represent a piece of music, since a variety of controls that influence the sound produced by a real instrument are not able to be represented in MIDI. However, MIDI benefits from being very compact, and offers much easier ways of extracting precise high-level musical information.

MIDI consists of a sequence of instructions called *MIDI messages*, which correspond to events or changes in a control parameter. Two of the most important types of messages are `Note On`, and `Note Off`, which respectively encode for the start and the end of a note. The timing in MIDI is controlled by a clock which emits ticks at regular intervals. The clock has a resolution defined in *parts per quarter note* (ppqn), and a greater resolution leads to the ability to express more precise rhythmic patterns. The most commonly used resolution is 24 ppqn, in which a half note (𝅗𝅥) corresponds to $24 \times 2 = 48$ ticks while an eighth note (♪) corresponds to $24/2 = 12$ ticks.

When MIDI needs to be encoded in a file, there are three standard MIDI file formats which can be used, numbered 0, 1, and 2. These formats differ in the way they encode different tracks, which can represent different voices or instruments. Format 0 files are made up of a single multi-channel track, Format 1 files have multiple tracks which all have the same meter and tempo, while Format 2 have multiple tracks with different tempos and meters. Format 1 files are the most commonly used.

Several technologies have been developed to work with MIDI files and perform a variety of analyses. Cuthbert & Ariza [10] introduced the `music21` Python library in 2010, which enables a wide range of processing operations, semantic analyses, and information retrieval, on MIDI files. We will be using the library throughout this project.

2.5 Datasets

In order to perform learning on symbolic music data, we will consider two datasets of interest within this project. First, we will use the Lakh MIDI Dataset [11], which contains a collection of 178,561 MIDI files on which annotations can then be added from other sources. For evaluation purposes in chapter 5, we also consider the smaller Bodhidharma MIDI Dataset [12], which contains 950 MIDI files labelled with their genre annotations from a set of 38 different labels.

Chapter 3

GenFastLAS: Scaling up FastLAS using Genetic Algorithms

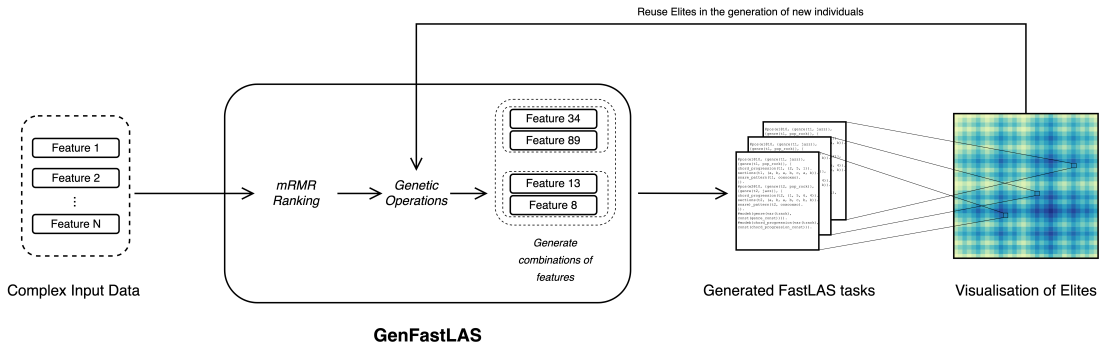


Figure 3.1: The architecture of GenFastLAS

As mentioned previously, Logic-based Learning systems offer a powerful way to approach Machine Learning tasks in a structured and explainable way, and can be very useful for research, creative, or educational purposes. Recent technologies, such as FastLAS [3], have been able to perform this kind of learning at scale, and have demonstrated great results. However, it still remains very difficult for those technologies to be applied to complex data, and to scale up correctly with respect to the number of features present. Other technologies, like Evolutionary and Genetic Algorithms, have been successfully used to explore complex feature spaces in a quick and efficient way, by emulating the behaviour of living organisms. A particular subclass of these algorithms, called Quality-Diversity (QD) algorithms, aims to perform this feature space exploration while favouring diversity within the solutions found. This search for diversity allows QD algorithm to be, as well, extremely useful in contexts of scientific research or for creative purposes. Although Logic-based Learning and Genetic Algorithms appear to be remarkably complementary, there has been no attempt yet, to the best of our knowledge, to unite their capabilities using the most recent state-of-the-art technologies for both.

We, therefore, propose to use a QD algorithm, called MAP-Elites [1], on top of the FastLAS system, to create the first system able to combine the abilities of both systems, and learn explainable and diverse solutions from complex data at scale. We call this system GenFastLAS. A simplified diagram of the architecture of the system is presented on Figure 3.1.

This chapter first focuses on the way we encode FastLAS tasks to be compatible with the MAP-Elites algorithm. We then describe the way in which we have modified the latter to correctly work with symbolic FastLAS tasks. We finally explore the different behaviour descriptors that can be used within our integration of MAP-Elites with FastLAS, and discuss how these can help GenFastLAS to be applied to interactive use cases.

3.1 Genetic encoding of a FastLAS task

In order to use a FastLAS task within the MAP-Elites algorithm, we first need to have a way of genetically encoding it in a format that MAP-Elites can process.

3.1.1 FastLAS Task

General definitions

As mentionned in Chapter 2, we denote a FastLAS task as $T = \langle B, M, E^+ \rangle$ with B our background knowledge encoded as an ASP program, M our mode bias, and E^+ our set of WCDPIs. Our mode bias is further defined as $M = \langle M_h, M_b \rangle$ with M_h (resp. M_b) a set of head (resp. body) mode declarations. Addtionally, each WCDPI e inside E^+ is further defined as $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$, with e_{id} an identifier for e, e_{pen} the penalty for not covering e, e_{pi} the partial interpretation associated with e, and e_{ctx} the context of e expressed as an ASP program. The partial interpretation is further defined as $e_{pi} = \langle e^{inc}, e^{exc} \rangle$, with e^{inc} (rep. e^{exc}) the inclusions (resp. exclusions) of the partial interpretation e_{pi}.

Definition 1. *We denote as $E_{ctx} = \bigcup e_{ctx}$ the set of all contexts used by all examples inside our task T. As such, $B \cup E_{ctx}$ refers to the entire knowledge base for our task.*

We are especially interested in the special case where our entire program $B \cup E_{ctx}$ contains a large number of predicates, that we will call *features*, and that can be considered as body mode declarations for the solution of the task T.

Example 1. *Anticipating the musical application of Chapter 4, we consider a FastLAS Task $T = \langle B, M, E^+ \rangle$ with:*

- $B = \emptyset$;
- $M = \langle M_h, M_b \rangle$, *and further:*
  - $M_h = \{$`genre(const(genre_const))`$\}$, *and*
  - M_b *defined dynamically by GenFastLAS (see later);*
- $E^+ = \{e_1, e_2\}$ *with two examples defined as:*
  - $e_1 = \langle e_{id_1}, e_{pen_1}, e_{pi_1}, e_{ctx_1} \rangle$, *with:*
    - $e_{id_1} =$ `e1`,
    - $e_{pen_1} = 10$,
    - $e_{pi_1} = \langle \{$`genre(jazz)`$\}, \{$`genre(pop_rock)`$\} \rangle$, *and*
    - $e_{ctx_1} = \{$`chord_progression((2, 5, 1)),`
      `sections((a, b, a, b, c, a, b)),`
      `snare_pattern(ooxooxxo)`$\}$
  - $e_2 = \langle e_{id_2}, e_{pen_2}, e_{pi_2}, e_{ctx_2} \rangle$, *with:*
    - $e_{id_2} =$ `e2`,
    - $e_{pen_2} = 10$,
    - $e_{pi_2} = \langle \{$`genre(pop_rock)`$\}, \{$`genre(jazz)`$\} \rangle$, *and*
    - $e_{ctx_2} = \{$`chord_progression((1, 5, 6, 4)),`
      `sections((a, b, a, b, c, b, b)),`
      `snare_pattern(ooxooxxo)`$\}$

This example considers the case where we want to learn how to classify the genre of a track based on its chord progression, and uses two example with ids `e1` *and* `e2` *where a "jazz" track and a "pop_rock" track are defined through three predicates.*

Dynamic generation of body mode declarations

Given a program with a large number of predicates, we would like to explore what subsets of those predicates are the most relevant in the solution of our task T, without having to solve T for all of those subsets.

For clarity reasons throughout the rest of this chapter, we define intuitively $p(m)$ as the function that generates all possible groundings of the mode declaration m. Conversely, we define $m(p)$ as the function that generates the mode declaration for the predicate p.

Definition 2. *Let P_b be the set of size N containing the body mode declarations $\{m_{b_1}, \ldots, m_{b_N}\}$ for all the features used inside our program $B \cup E_{ctx}$ for which we want to check the relevance as part of solution of the task T. Note that we assume in this work that there is only one possible body mode declaration per predicate, that is that $|m(p)| = 1, \forall p \in B \cup E_{ctx}$. This is, however, not a limitation per se, since adding support for more possible body mode declarations would simply result in a growth of the search space.*

We then consider that the set of body mode declarations M_b in our task T can potentially be made up of any subset of P_b. We have that $M_b \in \mathcal{P}(P_b)$ with $\mathcal{P}(P_b)$ the power set of P_b. This means that M_b can take $|\mathcal{P}(P_b)| = 2^N$ values, which means that the size of the set of all possibles values of M_b grows exponentially as the number of predicates in $B \cup E_{ctx}$ grows linearly. Due to limitations in the FastLAS algorithm, this means that the problem quickly becomes intractable as we expand the set of predicates that are contained in our program, or intuitively the number of features in our data. Genetic Algorithms are then a promising approach to explore the search space in a quick and efficient way, that also avoids having to compute all different possibilities.

Example 2. *Using the same task T as in Example 1, we have the set:*

$$P_b = \{\texttt{chord\_progression(const(chord\_progression\_const))},\ \texttt{sections(const(sections\_const))},\ \texttt{snare\_pattern(const(snare\_pattern\_const))}\}$$

as well as the different possible constants:

- `chord_progression_const((2, 5, 1))`, `chord_progression_const((1, 5, 6, 4))`
- `sections_const((a, b, a, b, c, a, b))`, `sections_const((a, b, a, b, c, b, b))`
- `snare_pattern_const(ooxooxxo)`

As such, we have that $M_b \in \{M_{b_1}, M_{b_2}, M_{b_3}, M_{b_4}, M_{b_5}, M_{b_6}, M_{b_7}, M_{b_8}\}$, with:

$$\begin{aligned}
M_{b_1} =& \{\},\\
M_{b_2} =& \{\texttt{chord_progression(const(chord_progression_const))}\},\\
M_{b_3} =& \{\texttt{sections(const(sections_const))}\},\\
M_{b_4} =& \{\texttt{snare_pattern(const(snare_pattern_const))}\},\\
M_{b_5} =& \{\texttt{chord_progression(const(chord_progression_const))},\ \texttt{sections(const(sections_const))}\},\\
M_{b_6} =& \{\texttt{chord_progression(const(chord_progression_const))},\ \texttt{snare_pattern(const(snare_pattern_const))}\},\\
M_{b_7} =& \{\texttt{sections(const(sections_const))},\ \texttt{snare_pattern(const(snare_pattern_const))}\},\\
M_{b_8} =& \{\texttt{chord_progression(const(chord_progression_const))},\ \texttt{sections(const(sections_const))},\ \texttt{snare_pattern(const(snare_pattern_const))}\}
\end{aligned}$$

Where M_b can take $2^3 = 8$ values as expected. Our task T will therefore have one of those sets of body mode declarations. Note that all of those sets of body mode declarations can be used as part of different tasks as well.

Looking at Example 2, it is clear that the search space of a FastLAS task grows exponentially as the number of predicates increases. This reinforces the motivation of using Genetic Algorithms in order to guide the search over such a large space.

3.1.2 Genetic encoding

As mentioned in Chapter 2, Genetic Algorithms approach the exploration of the search space with methods that aim to mimic natural genetic processes. As such, *individuals* inside a population differ by their *genotypes* (also sometimes called chromosomes), which are composed of various *genes* that can take different values inside a set of various *alleles*. Genetic operations, such as *crossover* or *mutation*, are then applied to the genotypes of selected individuals and act as a source of randomness. We then compute the *fitness* for each individual, which also encodes the likelihood of their genes being used in the generation of new individuals.

On top of this, QD algorithms introduce a *behaviour descriptor* for each individual, which describes the behaviour of an individual with respect to a set of features of interest. The added presence of this behaviour descriptor greatly increases the future interactivity potential of the algorithm, and the different potential values it can take are described later in Section 3.3. For the moment, we consider a naive implementation where the behaviour descriptor is a 2-dimensional vector whose value is set to a pair of coordinates generated from the genotype of the individual.

Definition 3. *We define our genotypic coding function* $g : \mathcal{P}(P_b) \mapsto \{0,1\}^N$ *as:*

$$g(M_b) = \{\chi_{M_b}(m_b) | m_b \in P_b\}, \quad \forall M_b \in \mathcal{P}(P_b)$$

where $\chi_{M_b}(m_b)$ *is the characteristic function of a given set* M_b*, which returns* 1 *if a given body mode declaration* m_b *is in* M_b*, and* 0 *otherwise. Our genotype therefore has* N *genes, which can always take the value of one of* 2 *alleles.*

Conversely, we have a genotypic decoding function $g^{-1} : \{0,1\}^N \mapsto \mathcal{P}(P_b)$*.*

Definition 4. *We define an* individual I *that directly maps to a FastLAS task* T*, as* $I = \langle G, H, F, BD \rangle$*, with:*

- $G \in \{0,1\}^N$*, the genotype of the individual, defined as* $g(M_b)$*,*
- H*, the optimal solution of the FastLAS task* T*,*
- $F \in \mathbb{R}$*, the fitness of the individual, computed as* $\mathcal{S}(H,T)$*, and*
- $BD \in \mathbb{R}^{N_{BD}}$*, the behaviour descriptor of interest for the task* T*.*

This allows us to encode all of the possible tasks $T \in \mathcal{T}_{OPL}$, with $\mathcal{T}_{OPL}$ the set of all possible ILP^{OPL}_{LAS} tasks, that is our population. Each task T has exactly the same background knowledge B, set of WCDPIs E^+, and set of head mode declarations M_h, but is made different through its set of body mode declarations M_b, and, incidently, its genotype G. Additionally, using the scoring function $\mathcal{S}$ to encode the fitness F of the individual allows us to customise the behaviour of the GenFastLAS algorithm as is required by the task.

Example 3. *Considering our example once again, we can create a set of genotypes of length 3 where:*

- *index 0 encodes for the inclusion of the body mode declaration for predicate* `chord_progression`*,*
- *index 1 encodes for the inclusion of the body mode declaration for predicate* `sections`*, and*
- *index 2 encodes for the inclusion of the body mode declaration for predicate* `snare_pattern`*.*

We consider indices to start from 0 and read from left to right. We can then encode different genotypes for the differents sets of body mode declarations of Example 2, as such:

$$\begin{aligned}
G_1 &= g(M_{b_1}) = \{0,0,0\} \\
G_2 &= g(M_{b_2}) = \{1,0,0\} \\
G_3 &= g(M_{b_3}) = \{0,1,0\} \\
G_4 &= g(M_{b_4}) = \{0,0,1\} \\
G_5 &= g(M_{b_5}) = \{1,1,0\} \\
G_6 &= g(M_{b_6}) = \{1,0,1\} \\
G_7 &= g(M_{b_7}) = \{0,1,1\} \\
G_8 &= g(M_{b_8}) = \{1,1,1\}
\end{aligned}$$

A visual example of encoding for a given set of body mode declarations is shown on Figure 3.2.

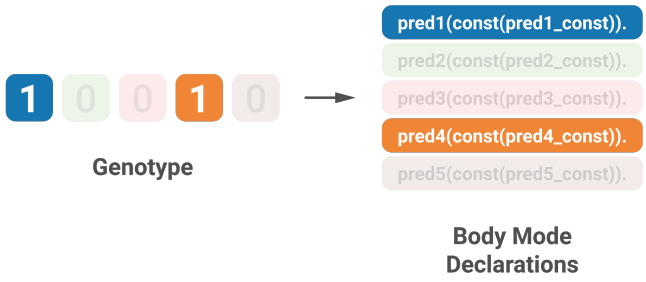


Figure 3.2: A genetic encoding of a possible set M_b with a set of predicates P_b of size $N = 5$

procedure MAP-ELITES ALGORITHM FOR GENFASTLAS (MODIFIED)
  $\mathcal{I} \leftarrow \emptyset$ ▷ Create an empty, N_{BD}-dimensional map of elites $\mathcal{I}$
  n_evals = 0 ▷ Keep track of the number of total evaluations
  while n_evals < max_evals **do** ▷ Repeat until we performed enough evaluations
    if $|\mathcal{I}| < I_{\text{min}}$ **then** ▷ Initialise with a minimal number of individuals
      Batch ← random_initialisation()
    else ▷ After initialisation, use computed genotypes
      Batch ← selection($\mathcal{I}$) ▷ Perform selection
      Batch ← variation(B) ▷ Perform variation
    for G **in** Batch **do** ▷ Evaluate all genotypes (parallelised)
      $BD, F, H \leftarrow$ evaluate_task(G) ▷ Evaluate FastLAS task with genotype G
      if $\mathcal{I}(BD) = \emptyset$ or $\mathcal{I}(BD) < F$ **then** ▷ If cell is empty or new fitness is better
        $\mathcal{I}(BD) \leftarrow \langle F, H, G \rangle$ ▷ Store fitness F, solution H, and genotype G in the map
        n_evals ← n_evals + 1
    if repeated no updates **then** ▷ If the map is not updated repeatedly, terminate
      break
  return archive of individuals(I)

Figure 3.3: Pseudocode adapted from the original MAP-Elites algorithm from [1]

3.2 Adapting the MAP-Elites algorithm to interact with FastLAS

We now focus on implementing the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm [1], to use as a way of efficiently exploring the hypothesis space of FastLAS tasks, while favouring diversity within the solutions that are learnt. Some modifications to the original algorithm are made to: (1) handle 1-dimensional binary genotypes, (2) use customised operators for discrete crossover and mutation, and (3) implement a heuristic to guide the generation of the first individuals. In the following subsections, we give an outline of the MAP-Elites algorithm, with an emphasis on the modifications that we performed. Our implementation is based on INRIA's Python3 MAP-Elites module [13]. An overview of our algorithm is given on Figure 3.3.

3.2.1 Step 1: Random initialisation

In order to kickstart the MAP-Elites algorithm, we need to evaluate the fitness of a few individuals, which can later serve as a basis for the natural selection process. The number of individuals that are considered in the first stage of the algorithm is determined through the `minimum_initialisation` parameter.

While the original algorithm performs this initialisation at random, we decide to use a heuristic in order to select the most appropriate genes first. This is done in the hope of starting the second step (Subsection 3.2.2) of the algorithm with high-performing individuals. Since mutation is performed at random in the second step of the algorithm, this does not prevent us from considering the entire search space, and thus still strive for diversity. We use the Minimum Redundancy

Maximum Relevance (mRMR) method [14] as a heuristic to select the most relevant and least redundant genes first. In order to do so, we first need to temporarily transform our complex learning task T into a simpler multi-featured multi-class classification problem, through a process that we describe here.

Converting the FastLAS task into a Multi-Class classification problem

We convert our FastLAS task into a Multi-Class classification problem that will solely help us compute the mRMR metric. This conversion is therefore temporary, and will be discarded once the learning is actually performed.

We first start by considering all of the examples $e \in E^+$, such that $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$ and $e_{pi} = \langle e^{inc}, e^{exc} \rangle$, as previously defined. For each example $e \in E^+$, we further consider every head mode declaration $m_h \in M_h$. Our goal is to define unique labels for the class labels of our data (corresponding to the different sets of values taken by head predicates), and for the features of our data (corresponding to the different sets of values taken by each body predicate)

Definition 5. *Given an example $e \in E^+$ and a head mode declaration $m_h \in M_h$, we denote as $Y_{(e,m_h)}$ the set of all possible groundings of the head mode declaration m_h that are compatible with the example e. Formally:*

$$Y_{(e,m_h)} = \{p_h | p_h \in p(m_h), \textit{such that } p_h \in e^{inc} \wedge p_h \notin e^{exc}\}$$

Additionally, we define $Y_e = \bigcup_{m_h \in M_h} Y_{(e,m_h)}$ as the combination of all head predicates p_h such that $p_h \in e^{inc} \wedge p_h \notin e^{exc}$ for a given example $e \in E^+$

Finally, we construct the set $Y = \{Y_e | e \in E^+\}$ as the set of all possible combinations of all head predicates p_h such that $p_h \in e^{inc} \wedge p_h \notin e^{exc}$ for each example $e \in E^+$.

Using Definition 5, we have Y as our "ground truth" for a given task T. This allows us to transform our complex learning task into a simple multi-class classification problem, by associating a unique label to each unique value of Y. We then have $|Y|$ class labels.

Definition 6. *We write as $\mathcal{Y} \in E^+ \mapsto Y$ the function that returns the class label of every example $e \in E^+$.*

Example 4. *We have defined two examples e_1 and e_2, and one head mode declaration m_h in Example 1. We therefore have:*

- $Y_{(e_1,m_h)} = \{$`genre(jazz)`$\}$, *and*
- $Y_{(e_2,m_h)} = \{$`genre(pop_rock)`$\}$.

This allows us to write $Y = \{\{$`genre(jazz)`$\}, \{$`genre(pop_rock)`$\}\}$, and we have two class labels `label_1` *and* `label_2` *such that:*

- $\mathcal{Y}(e_1) =$ `label_1`, *and*
- $\mathcal{Y}(e_2) =$ `label_2`.

We can then proceed with a similar approach to compute unique labels for every feature of our dataset.

Definition 7. *Given an example $e \in E^+$ and a body mode declaration $m_b \in P_b$, we denote as $X_{(e,m_b)}$ the set of all possible groundings of the body mode declaration m_b that are compatible with the example e. Formally:*

$$X_{(e,m_b)} = \{p_b | p_b \in p(m_b), \textit{such that } p_b \in e_{ctx}\}$$

We further define $X_{m_b} = \{X_{(e,m_b)} | e \in E^+\}$ as the set of all combinations of the body predicate p_b associated with the declaration m_b, such that $p_b \in e_{ctx}$ for each example $e \in E^+$.

Once again, we have X_{m_b} as the "ground truth" for the feature associated with the body mode declaration m_b in the data of our task T. Each feature with declaration m_b therefore has $|X_{m_b}|$ labels.

Definition 8. *We write as* $\mathcal{X}_{m_b} \in E^+ \mapsto X_{m_b}$ *the function that returns the categorical label of the feature associated with declaration* m_b*, for every example* $e \in E^+$*.*

Example 5. *Applying those definitions to our previous example, we have for example* e_1*:*

- $X_{(e_1,m_{b_1})} = \{$`chord_progression((2, 5, 1))`$\}$,
- $X_{(e_1,m_{b_2})} = \{$`sections((a, b, a, b, c, a, b))`$\}$, *and*
- $X_{(e_1,m_{b_3})} = \{$`snare_pattern(ooxooxxo)`$\}$.

And for example e_2*:*

- $X_{(e_2,m_{b_1})} = \{$`chord_progression((1, 5, 6, 4))`$\}$,
- $X_{(e_2,m_{b_2})} = \{$`sections((a, b, a, b, c, b, b))`$\}$, *and*
- $X_{(e_2,m_{b_3})} = \{$`snare_pattern(ooxooxxo)`$\}$.

We further have:

- $X_{m_{b_1}} = \{\{$`chord_progression((2, 5, 1))`$\}, \{$`chord_progression((1, 5, 6, 4))`$\}\}$,
- $X_{m_{b_2}} = \{\{$`sections((a, b, a, b, c, a, b))`$\}, \{$`sections((a, b, a, b, c, b, b))`$\}\}$, *and*
- $X_{m_{b_3}} = \{\{$`snare_pattern(ooxooxxo)`$\}\}$.

This allows us to have 2 labels for feature with declaration m_{b_1}*:* `mb1_label_1` *and* `mb1_label_2`*, 2 labels for feature with declaration* m_{b_2}*:* `mb2_label_1` *and* `mb2_label_2`*, and 1 label for feature with declaration* m_{b_3}*:* `mb3_label_1`*, such that:*

- $\mathcal{X}_{m_{b_1}}(e_1) =$ `mb1_label_1`,
- $\mathcal{X}_{m_{b_1}}(e_2) =$ `mb1_label_2`,
- $\mathcal{X}_{m_{b_2}}(e_1) =$ `mb2_label_1`,
- $\mathcal{X}_{m_{b_2}}(e_2) =$ `mb2_label_2`,
- $\mathcal{X}_{m_{b_3}}(e_1) =$ `mb3_label_1`, *and*
- $\mathcal{X}_{m_{b_3}}(e_2) =$ `mb3_label_1`.

mRMR Implementation

We now have $\mathcal{Y}$ as our class labels, and $\mathcal{X}_{m_b}, m_b \in P_b$ as our data features, and are able to compute the heuristic for each of those features. Following Zhao et al.'s [15] implementation of mRMR in Machine Learning, we implement the metric using the F-test Correlation Quotient (FCQ) approach.

Definition 9. *We define a scoring function* $f^{FCQ} : P_b \mapsto \mathbb{R}$*, which computes the mRMR score for the feature associated with a given mode declaration, as such:*

$$f^{FCQ}(m_b) = F(\mathcal{Y}, \mathcal{X}_{m_b}) / [\frac{1}{N} \sum_{m_{b'} \in P_b} \rho(\mathcal{X}_{m_{b'}}, \mathcal{X}_{m_b})]$$

Where $F \in (E^+ \times E^+) \mapsto \mathbb{R}$ *is the F-test scoring function, and* $\rho \in (E^+ \times E^+) \mapsto \mathbb{R}$ *is the Pearson correlation.*

Since our features are categorical, we encode them using Scikit-learn's [16] James-Stein Category Encoder. We can then compute $f^{FCQ}(m_b)$ *for every* $m_b \in P_b$ *and define an ordering* $\leq_{P_b} = \{\langle m_{b_1}, m_{b_2} \rangle, m_{b_1}, m_{b_2} \in P_b | f^{FCQ}(m_{b_1}) \leq f^{FCQ}(m_{b_2})\}$

Example 6. *Using our previous example, we have the following dataset:*

	$\mathcal{X}_{m_{b_1}}$	$\mathcal{X}_{m_{b_2}}$	$\mathcal{X}_{m_{b_3}}$	$\mathcal{Y}$
e_1	`mb1_label_1`	`mb2_label_1`	`mb3_label_1`	`label_1`
e_2	`mb1_label_2`	`mb2_label_2`	`mb3_label_1`	`label_2`

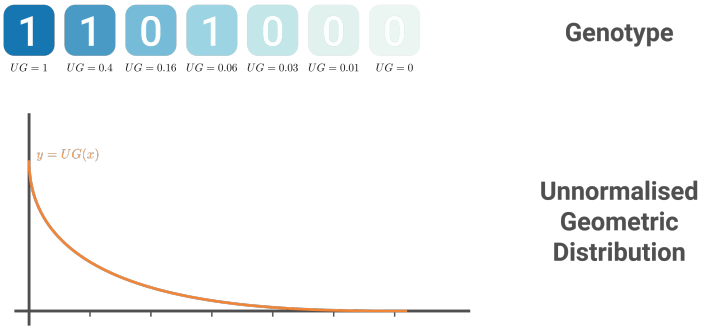


Figure 3.4: We select genes using a unnormalised geometric distribution

This, in turn, allows us to compute a mRMR score for each feature. We have:

- $f^{FCQ}(m_{b_1}) = 1$
- $f^{FCQ}(m_{b_2}) = 1$
- $f^{FCQ}(m_{b_3}) = 0.5$

We therefore have the ordering $\langle m_{b_1}, m_{b_2}, m_{b_3} \rangle$.

Random selection of the predicates

Now that we have a way of ordering features, we need to select them appropriately in the generation of our first genotypes. To do so, we make use of a *unnormalised geometric distribution* defined as such:

Definition 10. *We define an unnormalised geometric distribution UG as:*

$$UG_{(p_1,p_2)}(n) = \max(p_1^n - (1 - p_2), 0)$$

with $p_1, p_2 \in (0, 1)$ *two parameters and* $n \in (0, N)$ *the index of a feature in a genotype.*

In our algorithm, p_1 is represented by the hyperparameter `best_initial_features_tail` and p_2 by the hyperparameter `best_initial_features_weight`. Note that our function UG is bounded between 0 and 1 and therefore allows us to associate a probability of selection to every feature with index n, that decreases as n grows. Intuitively, p_1 (`best_initial_features_tail`) controls the rate of decrease of our probabilities, while p_2 (`best_initial_features_weight`) controls the maximum probability. A value of $p_2 = 1$ ensures that our best mRMR feature (with index $n = 0$) is always chosen.

By juxtaposing this distribution on top of our features ordered according to our previous ordering $\leq_{P_b}$ (see Figure 3.4), we can associate a probability of selection to each feature.

Example 7. *Given parameters* $p_1 = 0.6$ *and* $p_2 = 0.8$*, and by reusing our ordering of features* $\langle m_{b_1}, m_{b_2}, m_{b_3} \rangle$*, we can compute the probability of selecting each feature in our initial genotypes (we start indexing the genotype at index* 0*).*

- $UG_{(p_1,p_2)}(0) = 0.8$*,*
- $UG_{(p_1,p_2)}(1) = 0.4$*, and*
- $UG_{(p_1,p_2)}(2) = 0.16$*.*

We can then compute the probability of selecting a few genotypes from Example 3:

- *For* $G_7 = \{1, 1, 0\}$*, a probability of* $.8 \times .4 \times .84 = .27$*,*
- *for* $G_5 = \{1, 0, 0\}$*, a probability of* $.8 \times .6 \times .84 = .4$*, and*
- *for* $G_2 = \{0, 0, 1\}$*, a probability of* $.2 \times .6 \times .16 = .02$*.*

Finally, we perform this selection multiple times to construct a set of size BS (controlled by the hyperparameter `batch_size`) for which we can then evaluate the fitness.

3.2.2 Step 2: Fitness evaluation

To evaluate the fitness of an individual with a genotype $G \in \{0,1\}^N$, we first construct the FastLAS task $T = \langle B, M, E^+ \rangle$ by keeping B, E^+ and M_h the same as other tasks, but by setting M_b as the set of all body mode declarations that we want to consider. To do so, we use our genotypic decoding function from Definition 3. We then have $M_b = g^{-1}(G)$.

Once the fitness score F, the solution H, and the behaviour descriptor BD have been computed for an individual, we update the individual to have $I = \langle G, H, F, BD \rangle$, as previously defined. We then add this computed individual I to the *archive* A of all elites. The archive is a mapping of behaviour descriptors to the corresponding best genotype, with its associated fitness and solution.

We perform this evaluation for all individuals inside our batch at the same time, by parallelising the evaluation of the different FastLAS tasks associated with the individuals. We refer to the set of those FastLAS tasks ran in parallel as one FastLAS *job*.

Parallelising Tasks

To exploit the parallelisation opportunities offered by MAP-Elites, we use the open-source software HTCondor [17] to distribute FastLAS tasks on different machines over a network. HTCondor allows us to do so by creating a High-Throughput Computing (HTC) environment, within which we can easily spread the different FastLAS tasks of a single job, in order to compute the fitness for many individuals at the same time. When a FastLAS job is started, we distribute the N_{tasks} FastLAS tasks contained within it on different machines, and make use of the `minimum_wait` parameter to determine the number of tasks to wait for before moving to the next epoch. If `minimum_wait` is an integer, each job will wait for the completion of `minimum_wait` tasks (or N_{tasks} if it is smaller). If `minimum_wait` is a real value between 0 and 1, each job will wait for the completion of `minimum_wait`$\times N_{\text{tasks}}$. Once the wait is over, the results are collected by reading the output file of each task, and each individual is added to the archive following the process described previously. Additionally, results for pending jobs are collected at the end of each epoch, and the individuals are added to the archive. Although not implemented within this project, this parallelisation could also be done over the several CPU cores of a single machine. This is left for future work.

3.2.3 Step 3: Selection and genetic operations

Once the threshold of `minimum_initialisation` individuals has been reached (I_{min} in Figure 3.3), the MAP-Elites algorithm continues to fill the archive by a generation process that mimics *natural selection*.

Selection

We start by selecting two batches of BS individuals each to consider as *parents* for the generation of more individuals. If the hyperparameter `select_best` is set, we perform *stochastic uniform selection*, which selects individuals based on their previously computed fitness. Every individual I is associated a probability to be computed equal to:

$$\frac{Fitness(I)}{\sum_{I' \in A} Fitness(I')}$$

With $Fitness$ the function that returns the previously computed fitness of an individual. If the hyperparameter `select_best` is not set, we simply select the parents at random.

We then consider BS tuples of two parents, and we apply *crossover* to generate more individuals. We also randomly apply *mutation* in order to generate randomness.

Crossover

Crossover is a way of generating two new individuals from the combination of two parents. While different methods are available, we decide to use two-point crossover as it brings more linearity when working with discrete genotypes. To do so, we sample two indices $x_1, x_2 \in (0, N)$ at random such that $0 \leq x_1 \leq x_2 \leq N$. We then consider two children with the following genotypes:

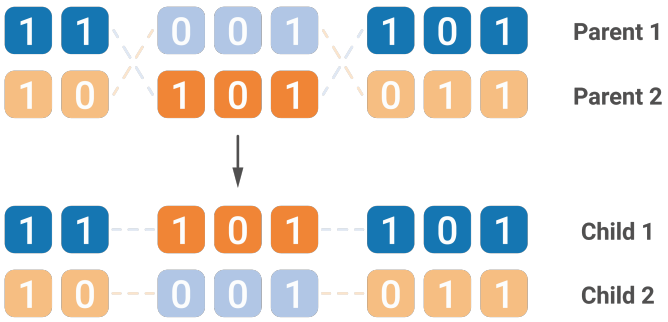


Figure 3.5: Two-point crossover applied to two different parent genotypes, resulting in two children genotypes

$$G_{\text{child}_1} = G_{\text{parent}_1}[0 : x_1] + G_{\text{parent}_2}[x_1 : x_2] + G_{\text{parent}_1}[x2 : N]$$
$$G_{\text{child}_2} = G_{\text{parent}_2}[0 : x_1] + G_{\text{parent}_1}[x_1 : x_2] + G_{\text{parent}_2}[x2 : N]$$

Where $G[a : b]$ refers to the selection of a part of the ordered set G located between indices a and $b - 1$ inclusive, and $+$ refers to the concatenation operator between two ordered sets.

Figure 3.5 shows the process behind the two-point crossover operator. We then randomly selects one of the child to be considered for evaluation.

Example 8. *Looking back at the genotypes defined in Example 3, we can simulate the effect of a crossover on the following selected parents:*

- *Parent 1 with genotype* $G_2 = \{1, 0, 0\}$
  - *Mode body:* $M_{b_2} = \{$`chord_progression(const(chord_progression_const))`$\}$
- *Parent 2 with genotype* $G_4 = \{0, 0, 1\}$
  - *Mode body:* $M_{b_4} = \{$`snare_pattern(const(snare_pattern_const))`$\}$

By randomly selecting $x_1 = 1$ *and* $x_2 = 2$*, we can construct two different children:*

- *Child 1 with genotype* $G'_1 = \{0, 0, 0\}$
  - *Mode body:* $M_{b_1} = \{\}$
- *Child 2 with genotype* $G'_2 = \{1, 0, 1\}$
  - *Mode body:* $M_{b_6} = \{$`chord_progression(const(chord_progression_const))`, `snare_pattern(const(snare_pattern_const))`$\}$

We observe that the resulting genotypes somehow combine parts of the parent genotypes, and we can intuitively see how such operations allow us to explore the hypothesis space efficiently.

Mutation

With a probability set by the `prob_mutation` hyperparameter, we perform mutation on one of the two parents (selected at random) and evaluate the fitness of the resulting individual. We apply a standard mutation by flipping each bit of G with a probability of $1/N$.

3.2.4 Picking out the best individuals

Once the algorithm stops, either due to a complete exhaustion of individuals to evaluate, or when the given number of evaluations has been reached, we need to have a way of evaluating the individuals of our archive. We might, for example, take the best performing solution, or select individuals who display specific behaviours. To do so, we use a held-out validation set, extracted at random from the training set, that was not used during the training. We then evaluate all of our individuals in our archive against this validation set, using the scoring function S for each individual. The returned score is then used for our selection of individuals of interest.

3.3 Exploring different behaviour descriptors

One of the main benefits of using MAP-Elites is the ability it offers, as a QD algorithm, of using different behaviour descriptors to both describe and explore the hypothesis space in a customisable way. As described in Subsection 3.1.2, using a behaviour descriptor allows us to specify the characteristics that we are interested in within the solutions learnt by FastLAS. The behaviour descriptor is also used to measure the distance between individuals, and is therefore used as part of the exploration of the hypothesis space, and to visualise the generated solutions.

Most of GenFastLAS has been developed around the naive implementation of having a bijection between the behaviour descriptors and genotypes of our individuals. This results in an iterative exploration of the hypothesis space, where each individual is mapped to a unique niche in the archive, therefore eliminating any competition at the niche level. However, while this implementation allows us to fully exploit the optimisation benefits of GenFastLAS, it disregards the potential for a precise control over the hypothesis space search brought by MAP-Elites. Additionally, selecting behaviour descriptors allows us to consider a feature smaller space than the entire genotypic space, and therefore allows visualisation in the case of very large feature sets.

In this short section, we explore the process that we developed to properly use a genotype as a behaviour descriptor, and correctly define a distance metric over the genotypic space. We then mention a few behaviour descriptors that can be used in the context of GenFastLAS, and how they can enhance Human-Model interactivity

3.3.1 Using the genotype as a behaviour descriptor

In order to use an individual's genotype as its behaviour descriptor, we need to be able to map any genotype G in our archive to a corresponding pair of 2D coordinates. However, this mapping cannot be done at random. As a matter of fact, to ensure the coherence of our behavioural space, we need to enforce the fact that similar individuals have coordinates that are close to each other. In order to measure the similarity between individuals, we use a distance metric that we apply to the genotypes of the individuals.

Definition 11 (Hamming Distance [18]). *Given two binary strings $u \in \{0,1\}^N$ and $v \in \{0,1\}^N$, both of length N, the* Hamming Distance $d(u,v)$ *is equal to the number of indices where both strings u and v differ. It can be proven that this distance is a correct distance metric.*

We decide to use the Hamming Distance applied on genotypes to measure the similarity between individuals. In our case, two genotypes differing by only one bit refer to two FastLAS tasks where only one body mode declaration has been added or removed. We can therefore argue that those two FastLAS tasks are indeed more similar than when taking tasks with completely different body mode declarations. Additionally, we can intuitively imagine that repeatedly adding or removing body mode declarations might lead to a continuous improvement or deterioration of the fitness of the solution, therefore ensuring linearity in our behavioural space. This distance metric also cognates with the way in which individuals are selected in our algorithm, since we apply a succession of bitwise operations.

Definition 12 (Karnaugh Maps [19], generalised). *A Karnaugh Map is a mapping of binary strings on a 2D grid, in which two adjacent strings only differ by exactly one bit.*

Example 9. *We can map the genotypes from Example 3 on a 3-variable Karnaugh map:*

$\{0,0,0\}$	$\{0,0,1\}$	$\{1,0,1\}$	$\{1,0,0\}$
$\{0,1,0\}$	$\{0,1,1\}$	$\{1,1,1\}$	$\{1,1,0\}$

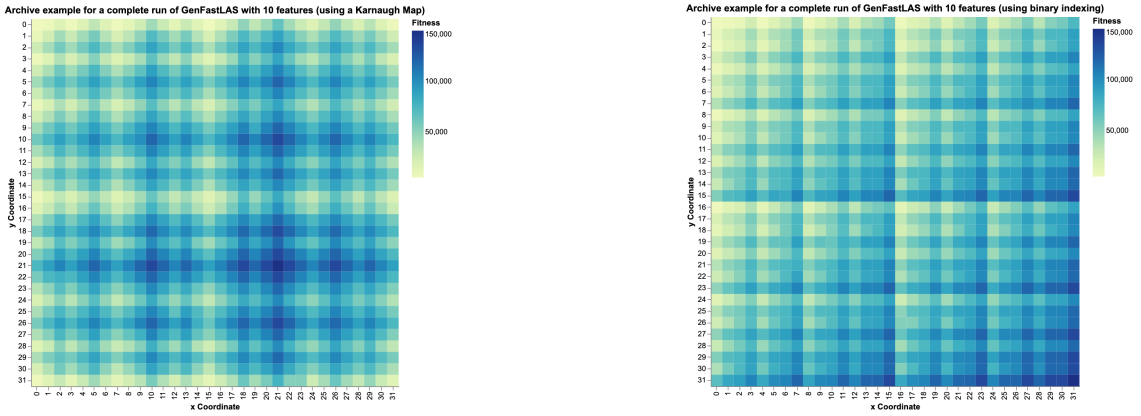


Figure 3.6: An example of 2D heatmaps generated for an archive computed by GenFastLAS for an ensemble of 10 features. The visualisation can use Karnaugh Maps (left) or naive binary indexing (right).

We therefore use our generalisation of *Karnaugh Maps* in order to map any genotype to 2D coordinates. Namely, we use the following recursive formula:

Definition 13. *We define a recursive function 2D_coordinates* : $\{0,1\}^X \mapsto (\mathbb{N}\times\mathbb{N})$ *that returns the 2D coordinates of any given genotype of length* X*, based on our generalised definition of Karnaugh Maps.*

$$2D\_coordinates(G) = \begin{cases} (0, G[0]) & \text{if } |G| = 1 \\ 2D\_coordinates(G[1:]) & \text{else if } G[0] = 0 \\ (\sqrt{2^{|G|}} - x(2D\_coordinates(G[1:])) - 1, & \\ y(2D\_coordinates(G[1:]))) & \text{else if } |G| \mod 2 = 0 \\ (x(2D\_coordinates(G[1:])), & \\ \sqrt{2^{|G|}} - y(2D\_coordinates(G[1:])) - 1) & \text{else if } |G| \mod 2 = 1 \end{cases}$$

with $G[0]$ *representing the most significant bit of the genotype* G*,* $G[1:]$ *representing the other bits, and* $x(\cdot)$ *(resp.* $y(\cdot)$*) the function that returns the* x *(resp.* y*) coordinate of a tuple of 2D coordinates.*

With a reasonable range of features, we can then visualise the archive as a 2D heatmap, on which the colour encodes for the fitness of the individual. Given that our genotype is of length N, we have a 2D heatmap of size $(\sqrt{2^N}, \sqrt{2^N})$ if N is even, and of size $(\sqrt{2^{N-1}}, \sqrt{2^{N+1}})$ if N is odd.

Figure 3.6 (left) shows an example of such a visualisation with a set of 10 features. We can see that the highest-performing individuals (referred to as *elites*) are present in different parts of the map, which suggests that the inclusion of different features translates in equally good solutions. This, in turn, can provide us with an interesting insight about the task at hand, which is another motivation for the use of the MAP-Elites algorithm. For a comparison, Figure 3.6 (right) shows the same archive visualised with a naive approach, where genotypes are indexed one after the other, from left to right and top to bottom. We can see that using Karnaugh Maps allows us to have much more meaningful clusters, compared to binary indexing, where linearity is harder to see.

3.3.2 Other behaviour descriptors

As mentioned previously, while using genotypes as behaviour descriptors still allows us to benefit from the optimisation brought by MAP-Elites, it fails to exploit properly the opportunities for model interaction given by the selection of custom behaviour descriptors. As a matter of fact, using genotypes as behaviour descriptors assigns one niche per genotype, and allows the efficient computation of the best genotypes with respect to the scoring function defined in the GenFastLAS task. However, we might sometimes be interested in exploring more dimensions at the same time than simply the scoring function, such as different metrics for explainability, or other domain-specific measures, and behaviour descriptors help us define those. Since behaviour descriptors

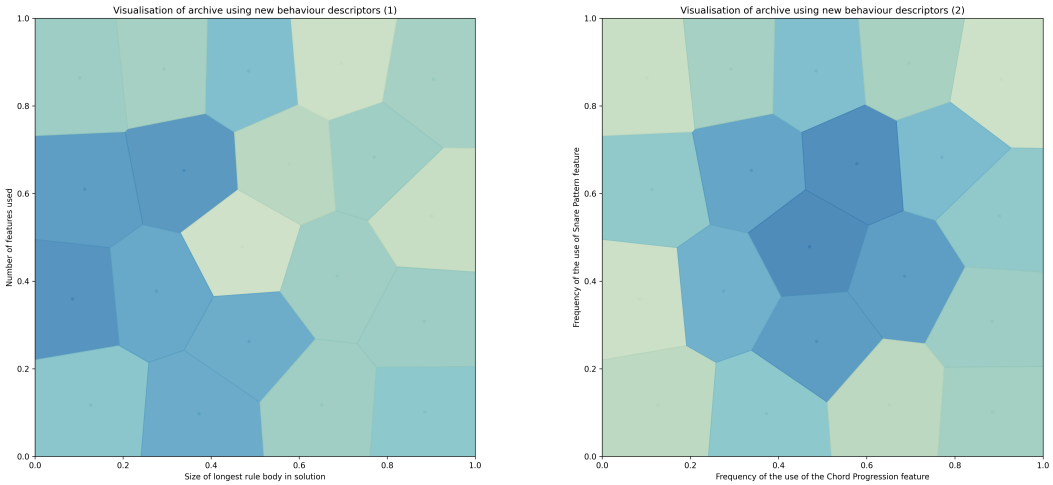


Figure 3.7: Two visualisation of a GenFastLAS archive using behaviour descriptors, and MAP-Elites with Voronoi Tesslation [2]. The figure on the left shows a visualisation using behaviour descriptors that measure the complexity of the solutions learnt. The figure on the right shows a visualisation using domain-specific behaviour descriptors for the domain of music.

dictate the way in which genotypes are evaluated and compared, modifying them during the learning in response to the current results can also bring a new aspect of interactivity within the learning system, such as in Alvarez et al. [20]. While GenFastLAS does not currently support such interactivity, we offer this as a future area of work, as we believe that it can bring a very interesting and novel dimension of interactivity within Logic-based Learning.

Behaviour descriptors can be computed from the solutions to the FastLAS tasks associated with specific individuals, or directly from the genotypes of those individuals. They can be defined in many ways and help measure the complexity of the solutions learnt, quantify domain-specific characteristics, and more. To improve explainability, behaviour descriptors can be chosen in such a way that they measure the complexity of the solutions learnt by the FastLAS tasks. Such descriptors can include the length of the learnt rule bodies, the number of features used per rule, or the variability of the constants used. Choosing those behaviour descriptors can help determine how the accuracy of the solutions varies with their complexity, which might be of value to specific users. Behaviour descriptors can also be used to compare solutions using domain-specific metrics, which can be of interest for research and educational applications. As a matter of fact, users of those domains might want to compare the solutions learnt based on additional metrics, visualise the current state of the learning process, and continue it with these new behaviour descriptors used as a competing factor. In the example of a music application, such metrics could include the number of unique sections, or chord progressions, used to define a genre, or the variability of the rhythmic patterns. Figure 3.7 shows two visualisation of GenFastLAS archives using two different sets of behaviour descriptors.

Chapter 4

Designing explainable features for symbolic music data

In the previous chapter, we introduced a new system which enables the automated and efficient learning of explainable knowledge from symbolic data. As discussed in Chapter 2, explainable features need to be inferable from symbolic data in order to maximise the explainability of the solutions learnt. In order to enable the application of GenFastLAS to the promising task of Symbolic Music Genre Clasification, and to be able to extract the most meaningful solutions from the task, we first need to have access to explainable symbolic music features.

In this chapter, we first discuss the current availability of explainable features for symbolic music data, and articulate the motivation for the design of new ones. We then present our set of eight novel and explainable symbolic music features, and describe the systematic approach for their computation. Both the explainability of our features and their efficiency when used within GenFastLAS are discussed in the next chapter.

4.1 Availability of explainable symbolic music features

4.1.1 Symbolic music dataset and preprocessing

As mentionned in Chapter 2, we make use of Raffel's Lakh MIDI Dataset (LMD) [11], which has been extensively used for research on symbolic music, and which contains 178,561 MIDI files that act as symbolic representation of music pieces. For the purpose of our task of Symbolic Music Genre Classification, we annotate those MIDI files using the genre mapping used initially by Schindler et al. [21], which contains genre labels extracted from the MSD Allmusic Genre Dataset (MAGD). In particular, we consider the top-MAGD subset, which includes the top 13 genres present in the dataset. These are: *Pop/Rock*, *Electronic*, *Country*, *RnB*, *Latin*, *Jazz*, *International*, *Vocal*, *Rap*, *New Age*, *Folk*, *Reggae*, *Blues*.

After filtering out any file without a genre label, we obtain a collection of 22,535 MIDI files.

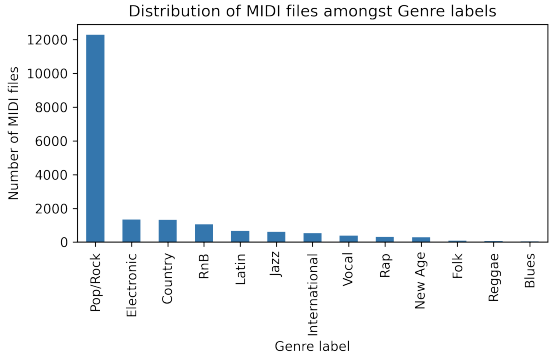


Figure 4.1: Distribution of the MIDI files amongst top-MAGD Genre labels

Genre label	Number of MIDI files
Pop/Rock	546
Electronic	530
Country	561
RnB	535
Latin	572
Jazz	598
Total	3,342

Table 4.1: Distribution of MIDI files for each of our 6 labels of choice, after preprocessing.

LMD is known to be heavily imbalanced, in favour of the *Pop/Rock* genre label. Figure 4.1 shows the distribution of the top-MAGD genre labels within LMD. In order to work with a balanced dataset, we select the top 6 labels and sample the same number of MIDI files from each of these labels. Namely, we sample the number of MIDI files that make up the limiting class (*Jazz*), giving us a total of 613 MIDI files per genre. We analyse each MIDI file using the `music21` library, and disregard any file that was not successfully analysed. We finally obtain a dataset with the distribution displayed on Table 4.1.

4.1.2 Existing symbolic music features

We now take a look at the features that we can extract from this dataset. As mentioned previously, in order to maximise the potential for explainable and interactive usages of these data, we aim to extract the most meaningful and easily intelligible features. This goes back to Clark and Brennan's previously cited work on the *"mutual knowledge"* required for human communication [6]. In order to ground this notion of *"mutual knowledge"* in the context of symbolic music, we need to first define what we mean the development of explainable features for symbolic music data. As with many human activities, music is surrounded by an extensive and formal theory, which is taught and communicated similarly accross multiple locations in the world. We therefore base our common ground of communication on this theory.

Definition 14 (following Clark and Brennan's theory [6])**.** *We describe a symbolic music feature as* explainable *if it embeds knowledge taken from music theory as it is found in litterature, and taught in academic institutions. The intuition for this definition is that, by using such features, machines are informed with a structured knowledge that is shared by humans, and which facilitates the communication with its users.*

In his PhD thesis, McKay introduces a corpus of 111 features that can be extracted from symbolic music data [22], which are all defined as part of one of seven categories: Instrumentation, Texture, Rhythm, Dynamics, Pitch Statistics, Melody, and Chords. To illustrate those features, we present one example of a feature for each category, together with descriptions directly cited from McKay's PhD thesis.

- **I-1 Pitched Instruments Present** (Instrumentation): "A feature vector with one entry for each of the 128 General MIDI Instruments. Each value is set to 1 if at least one note is played using the corresponding patch, or to 0 if that patch is never used."
- **T-1 Maximum Number of Independent Voices** (Texture): "Maximum number of different channels in which notes are sounded simultaneously."
- **R-1 Strongest Rhythmic Pulse** (Rhythm): "Bin label of the bin of the beat histogram with the highest magnitude."
- **D-1 Overall Dynamic Range** (Dynamics): "The maximum loudness value minus the minimum loudness value."
- **P-1 Most Common Pitch Prevalence** (Pitch Statistics): "Fraction of Note Ons corresponding to the most common pitch."

- **M-1 Melodic Interval Histogram** (Melody): "A feature vector consisting of the bin magnitudes of the melodic interval histogram."
- **C-1 Vertical Intervals** (Chords): "A feature vector consisting of the bin magnitudes of the vertical interval histogram.."

McKay's fascinating work has, therefore, allowed the generation of a large number of features which have all proven to be very powerful in different symbolic Music Information Retrieval tasks. However, since all of these features have been developed with the aim of using them within Support Vector Machines (SVMs), they suffer from the considerable drawback of being numerical values. Since music theory tends to be taught and communicated through an ensemble of much more hierarchical, categorical and structured concepts, we argue that having only numerical features intensely decreases the potential of these features to be used in explainable AI learning systems. We also argue that, since Logic-based Learning systems like GenFastLAS are generally designed to work with categorical features, we would like to be able to rely on this type of data to perform various symbolic Music Information Retrieval tasks, like our Symbolic Music Genre Classification task. As such, we attempt to design our own explainable and categorical features, based on music theory knowledge.

4.2 Our new features

Music is heavily based on meaningful and repeating patterns, due to its time-dependent characteristics. We believe that the presence and properties of those patterns greatly influence the genre classification of a track, and are strongly connected to the way humans naturally listen to and comprehend music. We therefore aim to develop a collection of explainable features, based on music theory knowledge that we encode manually.

As such, we extract patterns occurring on three different musical levels: at a high level, through repeated sections of a piece of music (see Subsection 4.2.1); on a harmonic level, through the repetition of successions of chords (see Subsection 4.2.2); and finally on a rhythmic level, through the repetition of motives played by the percussive instruments of a piece of music (see Subsection 4.2.3).

4.2.1 Feature 1: Sections

Sections, when discussed in the context of musical analysis, represent clearly-defined portions of musical content that get repeated throughout a piece of music. Sections can sometimes be part of compositional conventions, especially when it comes to musical genres. For example, there is an important amount of modern popular music based on the pattern *Verse-Chorus-Verse-Chorus-Bridge-Chorus*, while, in classical music, fugues generally follow a progression of *Exposition-Development-Recapitulation* [23]. While the names used to refer to those sections might differ based on the genre, they all refer to the same concept: the repetition of musical content that is similar. This similarity can be expressed through a multitude of musical concepts, like the key used, the melody played, or even the different rhythms present in the percussive parts.

Our section detection algorithm is heavily based on the Refrain Detecting (RefraiD) Method [24], but is simplified and adapted to work with symbolic music instead of raw audio.

Computing chroma vectors for measures

Following the RefraiD method, we start by computing a *chroma vector* to represent each measure in our music piece. Whereas RefraiD originally uses audio frames of a given size, we decide to use one measure as the atomic size in our representation. This is motivated by explainability reasons, since the measure is heavily used by humans as a representative unit when listening, studying or interpretating music. We also expect sections to be longer than one measure (generally at least 3 or 4 in most of the cases), which makes this choice coherent with our purpose.

Definition 15. *We define a 12-dimensional* chroma vector $\vec{v}(m)$ *for measure* m *as the vector containing the number of pitches played by all instruments for each pitch in measure* m*, normalised by the total number of notes in the measure. The 12 pitches used are taken from the modern music twelve-tone equal temperament (12-TET) standard and are:* C*,* Db*,* D*,* Eb*,* E*,* F*,* F#*,* G*,* Ab*,* A*,* Bb*, and* B*.*

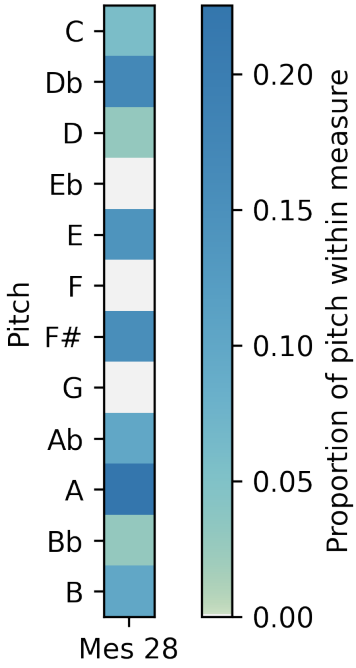


Figure 4.2: A visual representation for the *chroma vector* corresponding to measure 28 of the song *Charmless Man* by Blur

Example 10. *We analyse the sections for the song* Charmless Man *by Blur, which is present in our dataset under the ID* `38dce7e08508b65b15d07be37fd20ae2`.

Measure 28 of Blur - Charmless Man is for example made up of the following pitches:

C	*Db*	*D*	*Eb*	*E*	*F*	*F#*	*G*	*Ab*	*A*	*Bb*	*B*
4	*12*	*2*	*0*	*10*	*0*	*11*	*0*	*7*	*16*	*2*	*7*

which we feed into a column vector and normalise by the total number of pitches inside the measure to get the following chroma vector

$$\vec{v}(m_{28}) = \begin{bmatrix} 0.06 \\ 0.17 \\ 0.03 \\ 0.00 \\ 0.14 \\ 0.00 \\ 0.15 \\ 0.00 \\ 0.10 \\ 0.23 \\ 0.03 \\ 0.10 \end{bmatrix}$$

A visual representation of this vector is given on Figure 4.2.

Creating chromagrams to represent entire pieces

Now that we have a way of numerically representing measures, we can represent an entire piece of music through a sequence of chroma vectors corresponding to consecutive measures. We then have a matrix as a numerical representation of the piece, that we call *chromagram*.

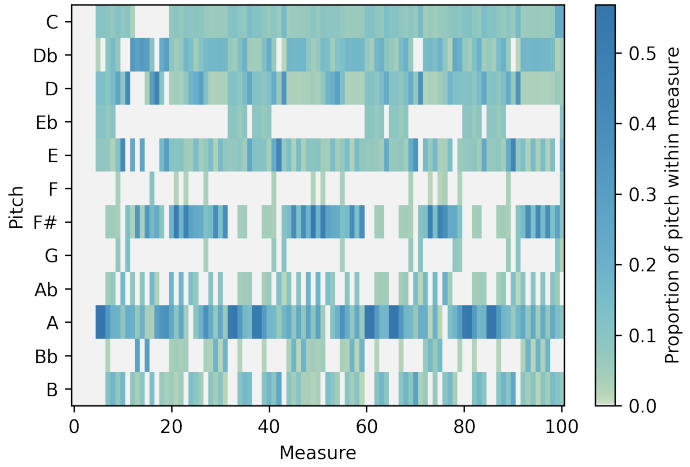


Figure 4.3: The chromagram for the song *Charmless Man* by Blur

Difference between chromagrams	Frobenius Norm
$\lvert M_2 - M_1 \rvert$	0.069
$\lvert M_3 - M_1 \rvert$	1.356

Table 4.2: Frobenius Norms for various differences between chromagrams for the song *Charmless Man* by Blur.

Definition 16. *We call* chromagram *of a musical piece $\mathcal{M}$ the matrix M of dimension $12 \times n$, with n the number of measures in $\mathcal{M}$, composed of the chroma vectors $\vec{v}(m)$ for every measure m in $\mathcal{M}$, such that:*

$$M = \left[\vec{v}(m_1), \ldots, \vec{v}(m_n)\right]$$

We additionally refer to portions of this chromagram as sections *and denote them as*

$$M[a:b] = \left[\vec{v}(m_a), \ldots, \vec{v}(m_{b-1})\right]$$

which represents the section from measures a to $b-1$ inclusive for the musical piece.

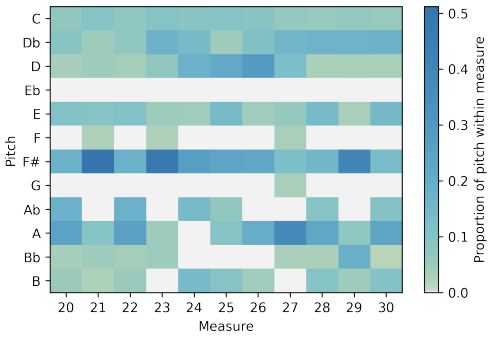
Figure 4.3 shows a visual representation of the chromagram for the song *Charmless Man* by Blur.

Measuring similarity between portions of the chromagram

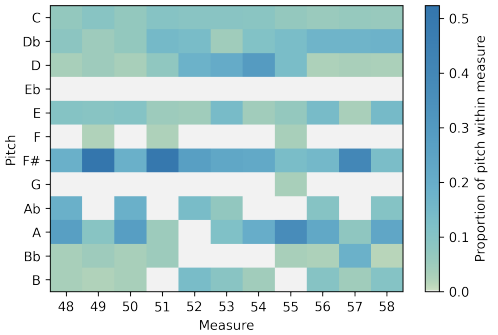
Given this new numerical representation for musical pieces, we can attempt to detect the presence and location of repeated sections. In order to do so, we first need to define formally what we mean by similarity in the context of symbolic music. For this matter, we simplify the RefraiD method and consider two measures to be similar when the proportion of pitches played within two measures is similar for each pitch. This allows us to measure the similarity between two measures through the Euclidean distance between the two corresponding chroma vectors. By generalising this concept, we can then compute the similarity between two sections through the *Frobenius Norm* of the difference between the two corresponding matrices.

Definition 17 (Frobenius Norm, from [25])**.** *The Frobenius Norm of a matrix A of dimension $m \times n$ is defined as:*

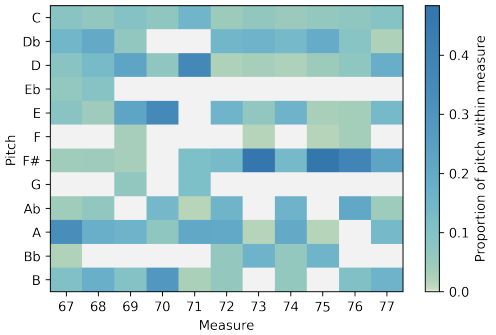
$$||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$$



(a) Section M_1 from measure 20 to 30 inclusive



(b) Section M_2 from measure 48 to 58 inclusive



(c) Section M_3 from measure 67 to 77 inclusive

Figure 4.4: Three sections of the song *Charmless Man* by Blur. The Frobenius Norm for the difference between M_1 and M_2 is 0.069, while the Frobenius Norm for the difference between M_1 and M_3 is 1.356.

```
procedure FindSections(M)                                   ▷ M is our chromagram
    N ← length(M)
    t ← 0.3                                                 ▷ We define a threshold for the norm
    S ← ∅                                                   ▷ S is our set of sections
    for F ← N/2 to 0 do                                     ▷ We consider all frame sizes
        for F1 ← 0 to N − 2F + 1 do
            for F2 ← F1 + F to N − F + 1 do
                M1 ← M[F1 : F1 + F − 1]                     ▷ We define the first frame
                M2 ← M[F2 : F2 + F − 1]                     ▷ We define the second frame
                if ||M1 − M2||F < t then
                    S = S + ⟨M1, M2⟩
        S ← remove_overlap(S)                               ▷ We remove overlapping sections
        if length(S) ≥ 2 then                               ▷ We keep the sections with the largest frame size
            break
    return S
```

Figure 4.5: Pseudocode for our chromagram traversal algorithm

Two sections are therefore said to be similar when the Frobenius norm of their difference is low. Figure 4.4 shows the visual representation of three different sections of the song *Charmless Man* by Blur. Section M_1 contains the measures 20 to 30 inclusive, section M_2 contains the measures 48 to 58 inclusive, and section M_3 contains the measures 67 to 77 inclusive. It is very intuitive visually to see that sections M_1 and M_2 are very similar, and that they should be classified as such. On the other hand, sections M_1 and M_3 do not seem to have much similarity, and the norm of their difference should be higher. The corresponding Frobenius norms are given on Table 4.2.

Automatically detecting sections

We can now traverse the chromagram of our musical piece in order to extract its most representative repeated sections. We do so through an iterative algorithm presented on Figure 4.5. Similarities are computed using the Frobenius norm, and different frame sizes for sections are considered. Most importantly, we consider all possible frame sizes ranging from half of the chromagram (resulting in two sections) to zero. However, we terminate once more than three similarities have been found, which results in us keeping the largest frame size that results in meaningful sections. This allows us to make sure that we consider the best similarities on a high explainable level, rather than on a per-measure basis, which would give us much less meaningful information.

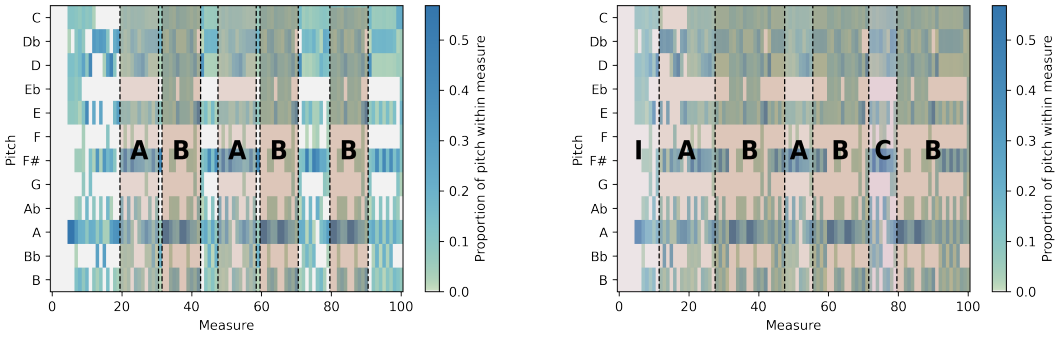


Figure 4.6: The chromagram for the song *Charmless Man* by Blur, overlayed with the sections detected by our algorithm (left), and with the sections as annotated by human users (right).

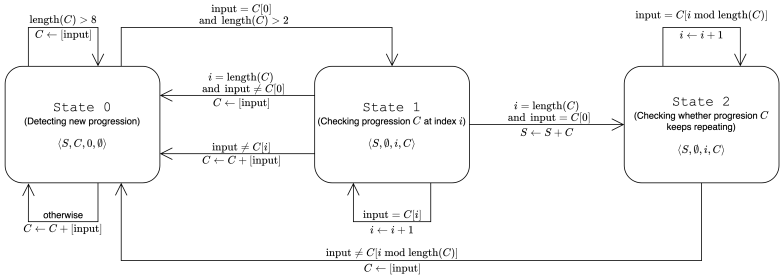


Figure 4.7: Diagram of the Finite State Machine used by our Chord Progression detection algorithm

Running the algorithm on Blur - *Charmless Man*, we detect 2 sections repeated as follows: *A - B - A - B - B*. According to the user-annotated lyrics from Genius.com [26], the song is made up of the following sections: *Verse 1 - Refrain - Chorus 1 - Refrain - Verse 2 - Refrain - Chorus 2 - Instrumental Break - Chorus 1 - Refrain*. We can further simplify this user-annotated sequence of sections by grouping the *Chorus* and *Refrain* sections together, due to their similarity in musical functionality. We then obtain a sequence which we can represent as *A - B - A - B - C - B*, by associating a different letter to each different type of section. We thus have a sequence of sections that is very similar to the one found by our algorithm. This comparison is shown on Figure 4.6, where the previously-mentioned user-annotated sequence of sections has been manually mapped to the different measures of the song. We can see that, although our algorithm does not detect the sections exactly as human users would annotate them, especially in terms of their boundaries, the order in which they appear is correct. Additionally, sections that appear only once (such as the introduction *I* and the instrumental break *C*) are not detected by our algorithm.

4.2.2 Feature 2: Chord Progressions

Similarly to musical sections, chord progressions offer an intuitive way to reason about the harmonical content of a piece of music, and, incidently, about its genre.

Extracting the sequence of all chords played in the piece

We first start by extracting the sequence of all the chords that are played in the musical piece. To do so, we first use the `chordify` functionality from the `music21` library, which allows us to reduce the multiple parts within our symbolic representation into one single succession of chords.

From there, we extract the *root* of each chord, which represents the most *harmonically important* note within that chord. This allows us to work with one value per chord, which we can assemble into one long string representation of the harmony of our music piece. We remove all consecutively identical values in order to reduce the redundancy and the length of our representation.

Example 11. *We look at the song* Hey Jude *by The Beatles, with ID* `2d370413bab5b0bd358ebd08-cecae503`. *Retrieving the root note for each chord gives us the following chord sequence for the entire music piece:*

F - C - F - Bb - F - C - F - C - F - Bb - F - C - F -
Bb - A - G - F - E - G - C - F - F - Bb - A - G - F -
E - G - C - F - C - F - C - F - Bb - F - C - F - Bb -
A - G - F - E - G - C - F - Bb - A - G - F - E - G - C -
F - C - F - C - F - Bb - F - C - F - Eb - Bb - F - Eb -
Bb - F - Eb - Bb - F - Eb - Bb - F - Eb - Bb - F - Eb -
Bb - F

We can intuitively see some repetition, which we aim to detect through our feature extraction algorithm.

Detecting patterns using a Finite State Machine

We then detect patterns in our string representation using a Finite State Machine, represented on Figure 4.7. The goal of this process is to find patterns of consecutive chords that get repeated

throughout the music piece. To do so, we traverse our string representation from beginning to end, keeping track of whether we are in a new sequence, or whether we are in a sequence that has been previously played. Our Finite State Machine therefore has 3 states:

- `State 0` attempts to construct a new chord progression from the sequence that is being traversed. It keeps track of which chords have been played (with a limit of eight chords), and detects when the first chord has been reached again. In this case, it moves to the next state.
- `State 1` checks whether the repeating sequence is exactly similar to a previously played chord progression, before adding it to the set of chord progressions. If the progression varies in some way, it does not add it and moves back to the previous state. If the progression is exactly similar, it adds it to the set and moves to the next state.
- `State 2` simply waits for the occurrence of a new chord progression. If the current chord progression keeps repeating, it simply stays in that state. If a new one is detected, it moves back to the initial state.

Example 12. *Running our algorithm on the chord sequence extracted from Example 11, we detect the following sequences:*

F - C - F - Bb - F - C - F - C - F - Bb - F - C - F - Bb - A - G - F - E - G - C - F - F - Bb - A - G - F - E - G - C - F - C - F - C - F - Bb - F - C - F - Bb - A - G - F - E - G - C - F - Bb - A - G - F - E - G - C - F - C - F - C - F - Bb - F - C - F - Eb - Bb - F - Eb - Bb - F - Eb - Bb - F - Eb - Bb - F - Eb - Bb - F - Eb - Bb - F

We can notice that the middle section (chorus) is not detected as a chord progression, due to the mishandling of "passing chords" by our algorithm. This happens due to the fact that `State 0` *will disregard the current chord progression when it grows too long. Handling "passing chords" better is left for future work.*

We then extract the two progressions:

- *F - C - F - Bb - F - C*
- *F - Eb - Bb*

Erasing key-dependency by extracting chord degrees

Now that we have our chord progressions, we need to be able to compare them between musical pieces. The issue that arises is that two musical pieces that have the same chord progressions might have different encodings using our system, due to their differing keys. A solution for this, however, is to encode our extracted chord progressions into a progression of *chord degrees*. These refer to the "harmonic location" of the chord within the key of the musical piece. We do so using the `music21` library.

Example 13. *In our example, the key of the musical piece is analysed as being F major. We can then map each chord progression into the following chord degrees:*

- *F - C - F - Bb - F - C:* `1 - 5 - 1 - 4 - 1 - 5`
- *F - Eb - Bb:* `1 - 7 - 4`

This therefore allows us to have a key-agnostic representation of chord progressions, which we can then use inside our learning task.

4.2.3 Feature 3: Drum Patterns

Finally, we aim to extract similarities on a *rhythmic* level, and detect repeated patterns played by different percussive sounds. This is arguably an easier task than for the previous two features, since we only consider a small subset of 47 MIDI notes, namely the ones defined by the specifications from the MIDI Manufacturers Association [27] as belonging to percussive instruments.

Extracting the rhythm of each measure

Similarly to the Sections feature describe in Subsection 4.2.1, we first start by constructing a representative vector for each measure within the musical piece. In the context of rhythmic patterns, however, we only consider the 47 MIDI pitches previously mentioned, that are played by percussive instruments only.

Definition 18. *We define the* rhythm vector $\vec{rv}(m)$ *for measure* m *as the number of percussive pitches played by all instrument for each percussive pitch in measure* m*. We disregard every percussive pitch that is never played in the musical piece, and only consider the* p *different pitches that are played. The vector* $\vec{rv}(m)$ *is therefore of size* p*, such that* $\vec{rv}(m) \in \mathbb{N}^p$*.*

We then construct the rhythm matrix *of a musical piece* $\mathcal{M}$ *as the matrix* RM *of dimension* $p \times n$*, with* n *the number of measures in* $\mathcal{M}$*, composed of the rhythm vectors for all measures in* $\mathcal{M}$*, such that:*

$$RM = \left[\vec{rv}(m_1), \ldots, \vec{rv}(m_n)\right]$$

Detecting repeating patterns

This allows us to retrieve a single vector of rhythmic content for every measure within a musical piece, and construct the ensemble of all of these vectors. Given the limited subset of pitches, we only consider exact patterns, which means that we only look for exact similarities between measures. We do so by, first, only selecting vectors with RM that appear for more than 10% of the musical piece, and, then, by constructing a set of distinct rhythmic patterns.

Definition 19. *We define our set of rhythmic patterns* $RP(M)$ *as the set of all distinct rhythm vectors that appear within* M *for more than 10% of the musical piece.*

Example 14. *We now look at the song* Waiting On the World to Change *by John Mayer, with ID* `2870e85f3de743e361644acc9a5f0120`*. Our algorithm detects one rhythm vector that appears more than 10% of the musical piece, represented here:*

$$RP(M) = \left[\vec{rv}\right] = \left[\begin{bmatrix} 0 \\ 2 \\ 6 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 11 \\ 0 \\ 16 \\ 0 \end{bmatrix} \right]$$

Where the indices of each row (from top to bottom) correspond to the following MIDI percussive pitches: `27`*,* `28`*,* `36`*,* `40`*,* `43`*,* `44`*,* `46`*,* `49`*,* `51`*,* `52`*,* `54`*, and* `57`*.*

Extracting an explainable representation

Now that we have our different repeating patterns, we can extract a human-readable representation out of them. We do so by creating a string from the sequence of percussive hits played over time for each percussive pitch class. We start by grouping MIDI percussive pitches into 6 different groups of sounds: *bass drum*, *snare*, *toms*, *hi-hats*, *cymbals*, and *others*. This mapping is summarised in Table 4.3.

For each of these groups, and for each pattern, we create a string representing each eighth of note in the measure, and mark whether a note is played in the duration of this eighth of note or not. This resolution of a 1/8th of note (♪) is decided arbitrarily, but, through experimentation, seems to offer a good compromise between precision and feature size.

Percussive Sound	MIDI Notes
Bass Drum	35, 36
Snare	37, 38, 39, 40
Toms	41, 43, 45, 47, 48, 50
Hi-Hats	42, 44, 46
Cymbals	49, 51, 52, 53, 55, 56, 57, 58, 59
Others	Any other in the range 35-81

Table 4.3: Mapping between our different classes of percussive sounds of interest, and the MIDI notes that they are associated to

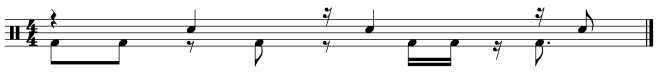


Figure 4.8: Sheet music for the rhythmic pattern of *Waiting On the World to Change* by John Mayer

Example 15. *For the pattern represented by the rhythmic vector from Example 14, we obtain the following* Bass Drum *and* Snare *patterns:*

- Bass Drum*:* `xxoxoxxo`
- Snare*:* `ooxoxoox`

The sheet music for this rhythmic pattern, extracted from the MIDI file, is shown on Figure 4.8.

Chapter 5

Evaluation

We now proceed to evaluate the different components introduced in this thesis, and on different aspects, by adapting different methods and metrics for each of these aspect. We start by evaluating GenFastLAS when applied to Symbolic Music Genre Classification, first compared with FastLAS, then with various baselines. We then move on to evaluate GenFastLAS in isolation, within the new context of a task traditionally approached using FastLAS. We finally evaluate the novel symbolic music features we developed, and study their explainability.

Throughout this chapter, we will try to answer the following questions:

(1) Does GenFastLAS bring an improvement in speed compared to FastLAS, and if so, how much?

(2) Does GenFastLAS bring an improvement in accuracy compared to FastLAS, and if so, how much?

(3) Is there any trade-off between speed and accuracy within GenFastLAS?

(4) Is there any trade-off between data efficiency and accuracy within GenFastLAS?

(5) Is the performance of our GenFastLAS system dependent on the number of examples to consider? If so, to what extent?

(6) Is the performance of our GenFastLAS system dependent on the number of features? If so, to what extent?

(7) How explainable are our new symbolic music Features, and how do they compare with existing jSymbolic features?

In order to measure aggregated metrics for multi-class classification tasks, we will often use *micro-averages*, which measure the precision of the aggregated contributions of all classes by assigning one-hot encodings to every class, and computing metrics in the same way as a binary classification.

5.1 Evaluating Symbolic Music Genre Classification using GenFastLAS

5.1.1 Comparing with FastLAS

We start by evaluating our Symbolic Music Genre Classification system using GenFastLAS, and compare it to using FastLAS alone. In this subsection, and for the purpose of being able to run multiple experiments in a reasonable amount of time, we will be working with a smaller subset of the dataset described in Subsection 4.1.1. Namely, we work with only 2 classes, *Pop/Rock* and *Jazz*.

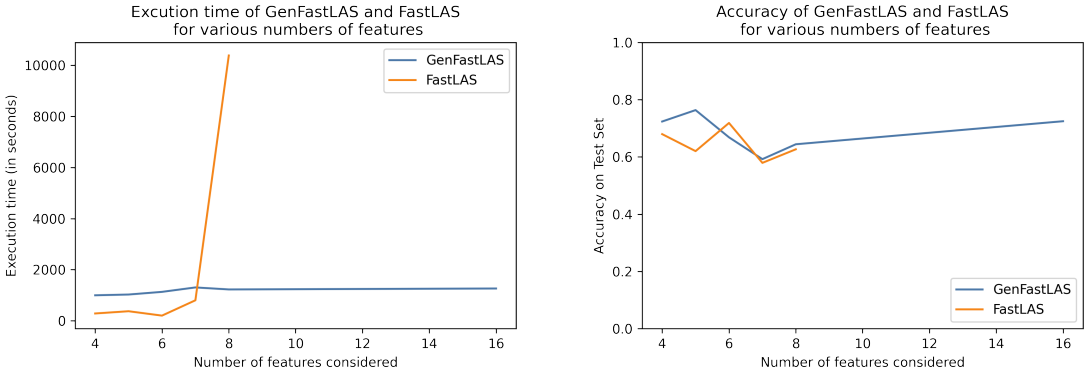


Figure 5.1: Comparison of execution time (left) and accuracy (right) of both GenFastLAS and FastLAS when applied to our Symbolic Music Genre Classification task for various numbers of features. Data is missing for FastLAS runs after 8 features, since the algorithm times out after 24 hours.

Improving Speed and Accuracy

The main improvement brought by GenFastLAS over FastLAS is the automated and efficient search that it provides of the hypothesis space. This removes the manual task of running multiple experiments and varying the body mode declarations every time, which is required when using FastLAS. Although it is difficult to measure this gain of practicality, we can nevertheless observe how both GenFastLAS and FastLAS perform on the same learning task with various numbers of features. For this purpose, we use a dataset of 100 datapoints, with 50 *Pop/Rock* tracks, and 50 *Jazz* tracks. We perform a stratified 5-fold cross-validation, and use 80 datapoints for training, and 20 datapoints for validation for each fold. Additionally, following the process described in Subsection 3.2.4, we hold out 10% of our training set when using GenFastLAS in order to select the best individual at the end of the training. This means that, in our comparison, GenFastLAS is trained on a slighly smaller training set than FastLAS within each fold.

We use our 8 novel features described in Section 4.2, which we complement with the 97 `jSymbolic` features offered by `music21`, that all correspond to different features from the previously quoted work by McKay [22]. We then obtain an entire corpus of 105 features. However, we only select a subset of them using our mRMR ranking described in Subsection 3.2.1.

Figure 5.1 shows the speed and accuracy of both GenFastLAS and FastLAS when applied to the same task, for various numbers of features. The speed is measured as the total time taken by both algorithms to perform the task, including all 5 folds, and the accuracy is computed as the average validation accuracy amongst all folds. The features are selected as the best mRMR features for both GenFastLAS and FastLAS. We can notice that, while GenFastLAS is slower than FastLAS for a small number of features (up to 5), it keeps the same speed with respect to the increase of the number of features, and quickly becomes faster than FastLAS once the number of features grows (from 7 features). We explain this behaviour as an overhead when using GenFastLAS for a manageable number of features. Since GenFastLAS needs to run a few iterations of FastLAS for every run, it needs more time than a single FastLAS task. However, the benefit of using GenFastLAS is exhibited when increasing the number of features since FastLAS will always attempt to use every single feature, while GenFastLAS will dynamically adapt the body mode declaration of the programs to find the optimal solution. We can also realise that accuracy is similar for both approaches, with GenFastLAS slightly overperforming FastLAS in most of the runs. While this difference is possibly not significant, we can hypothesise that the strategy that GenFastLAS uses in order to find the most relevant set of features to use can lead to better accuracies.

Figure 5.2 shows the AUC-ROC curves for both GenFastLAS and FastLAS when applied to our Symbolic Music Genre Classification task using 5 features. The higher ROC curve for GenFastLAS shows that GenFastLAS can reach a better performance by finding an efficient combination of features, whereas FastLAS reaches a lower performance when trying all features at the same time.

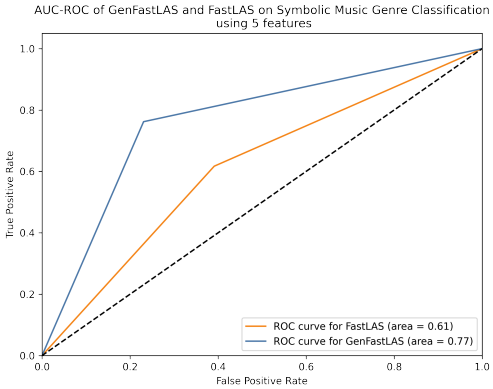


Figure 5.2: Comparison of the AUC-ROC of both GenFastLAS and FastLAS when applied to our Symbolic Music Genre Classification task using 5 features.

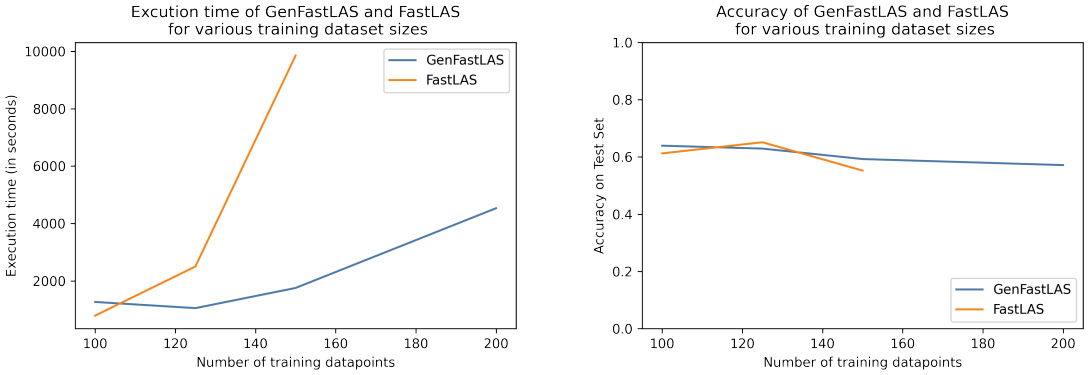


Figure 5.3: Comparison of execution time (left) and accuracy (right) of both GenFastLAS and FastLAS when applied to our Symbolic Music Genre Classification task for various sizes of training dataset. FastLAS timed out for more than 150 datapoints.

These observations answer questions (1), (2) and (6), by showing that GenFastLAS can outperform FastLAS with respect to both speed and accuracy, when the number of features becomes large enough.

Scaling up with respect to the number of examples

We now consider the case where the number of examples increases, and measure the difference in performance between the use of GenFastLAS and FastLAS. To do so, we use a set of 7 features, since this is the situation where GenFastLAS and FastLAS are the most similar in the amount of time they take to run and in their accuracy, as can be observed on Figure 5.1. We run the same experiment using a varying number of examples, always including the same amount of each genre label in order to preserve the original distribution.

Figure 5.3 shows that both GenFastLAS and FastLAS are unable to scale properly with respect to an increase of the number of examples. While GenFastLAS is still able to run for larger datasets, the execution time needed highly increases as we add training datapoints. This is because GenFastLAS does not bring much improvement in the exploration of the example space over FastLAS, but only in the exploration of the hypothesis space. As mentioned before, we leave out scaling in this dimension for future work. We can, however, notice that GenFastLAS seems to deal with an increase in this dimension in a better way, which we explain by the fact that the algorithm will naturally steer towards smaller solutions, including only a low number of relevant features, which takes less time to compute. Additionally, the accuracy only decreases very little when adding more training datapoints. This suggests that, while there is a slight trade-off between

Model	F1 (micro-averaged)	Execution time (in seconds)
GenFastLAS	0.2460	58,798
Bodhidharma	0.3154	58
MusicBERT	0.7186	74,339

Table 5.1: Comparison of micro-averaged F1 scores and execution time for the three systems, when applied to the full Lakh MIDI Dataset (apart from GenFastLAS)

speed and accuracy within GenFastLAS, it is in fact minimal.

These observations answer questions (3) and (5), by showing that there is a minimal trade-off between speed and accuracy within GenFastLAS, and by showing that the performance of GenFastLAS is dependent on the number of examples included in the example space, but to a much lower degree than FastLAS.

5.1.2 Comparing with other systems

Using the entire dataset

We now aim to compare GenFastLAS to other existing systems when applied to the task of Symbolic Music Genre Classification, using a full dataset. We consider two other systems, listed here:

- McKay's *Bodhidharma* system [22], which uses a SVM to classify symbolic music using the jSymbolic features, that we also use on top of our own features within our system, and
- Zeng et al.'s *MusicBERT* [28], which uses Bidirectional Encoder Representations from Transformers (BERT) to classify symbolic music, using a novel Symbolic representation, called OctupleMIDI. It is to be noted that this model is pre-trained, which needs to be considered when observing its really high performance throughout this evaluation.

We train both baselines on the entire Lakh MIDI dataset, although after filtering our 6 genres of interest as mentionned in Subsection 4.1.1. This is done in order to be able to compare the results with our system, which only uses those 6 labels. After filtering on those 6 labels, the dataset contains 22,535 datapoints, which are used by both the Bodhidharma system and MusicBERT. GenFastLAS, on the other hand, is only trained using a subset of this dataset. As we have just shown, GenFastLAS is able to scale up with respect to a growth of the hypothesis space, but, just like FastLAS, it does not scale well with an increase of the number of examples. As such, we split our dataset by using 150 training examples in each fold when running GenFastLAS.

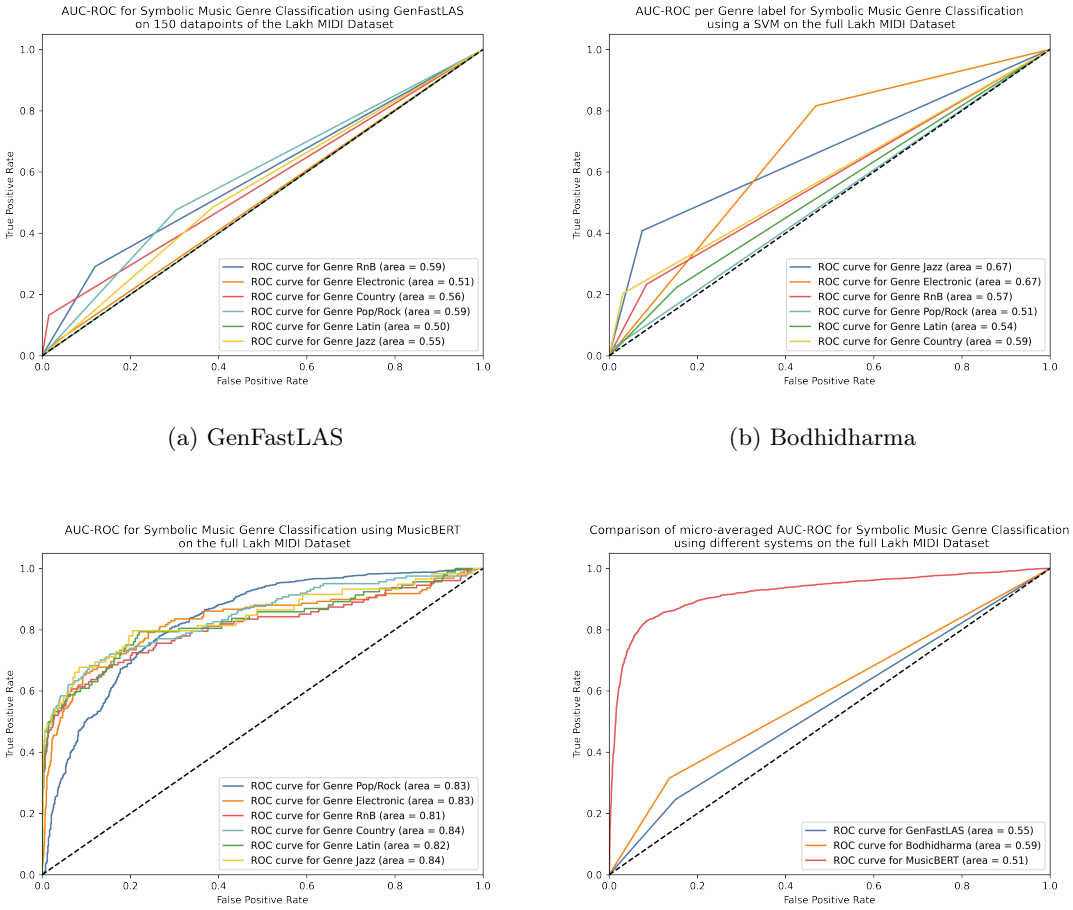
We run our system with a maximum of 30 features, a batch size of 40, a minimum waiting of 25% of the tasks, and a minimum initialisation of 40 individuals. We also run the Bodhidharma system on a Linear Support Vector Classification (SVC) with 10,000 iterations, l_2 penalty and a squared hinge loss. Finally, we run MusicBERT with the default parameters for the $\text{BERT}_{\text{small}}$ setting.

Figure 5.4 shows the AUC-ROC curves for each three systems when applied to our Symbolic Music Genre Classification task, and Table 5.1 presents the execution time and accuracy of each. It can be observed that GenFastLAS performs much less well than all other systems when used in this specific context. We explain this behaviour through the very limited set of examples that GenFastLAS is able to process for its training, compared to the bigger training datasets of both other baselines. Additionally, the very high F1 score for MusicBERT is also due to the fact that it is a pre-trained model.

Displaying Data Efficiency

We now aim to evaluate how our baselines perform when applied to the same amount of data as GenFastLAS. We run the same experiment after limiting the number of training datapoints to 150 for all systems.

Table 5.2 presents the execution time and accuracy of each system when applied to the same amount of data in our Symbolic Music Genre Classification task. Once again, MusicBERT is previously pretained, which explains its low execution time in our context. Figure 5.5 also shows the new AUC-ROC curves obtained for each three systems in this context. We can observe that



(a) GenFastLAS

(b) Bodhidharma

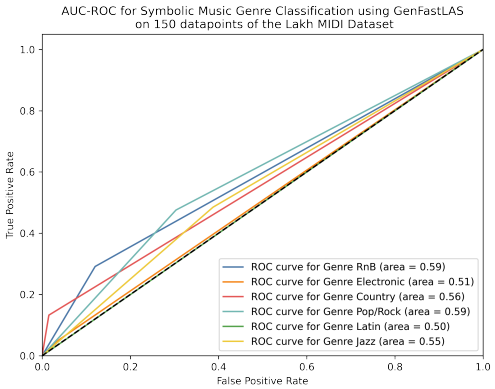
(c) MusicBERT

(d) Comparison

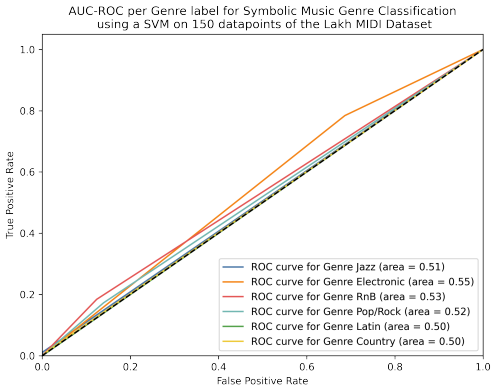
Figure 5.4: Comparison of AUC-ROC curves for the three systems trained on the full Lakh MIDI Dataset (apart from GenFastLAS). The bottom-right figure shows the comparison of micro-averaged AUC-ROCs.

Model	F1 (micro-averaged)	Execution time (in seconds)
GenFastLAS	0.2460	58,798
Bodhidharma	0.1857	23
MusicBERT	0.0570	982

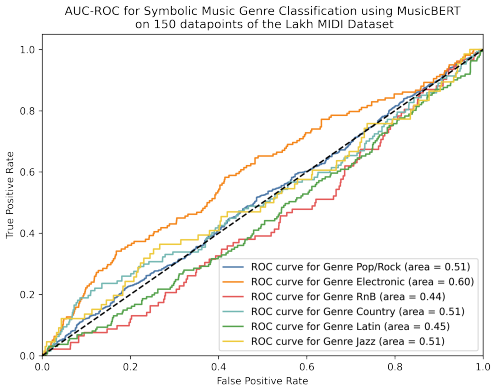
Table 5.2: Comparison of micro-averaged F1 scores and execution time for the three systems, when applied to 150 datapoints of the Lakh MIDI Dataset.



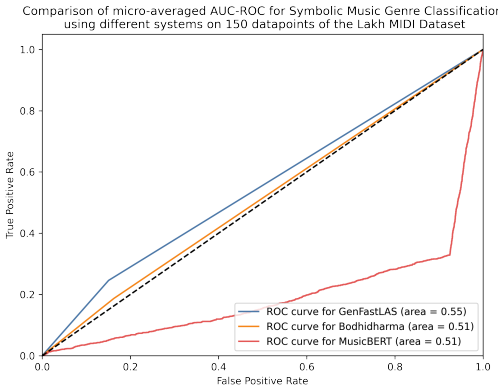
(a) GenFastLAS



(b) Bodhidharma

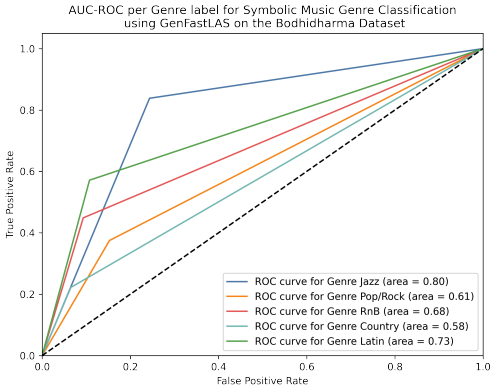


(c) MusicBERT

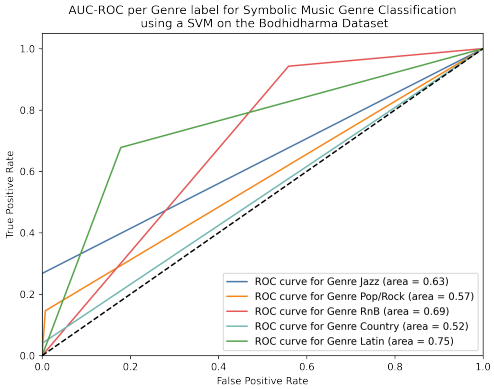


(d) Comparison

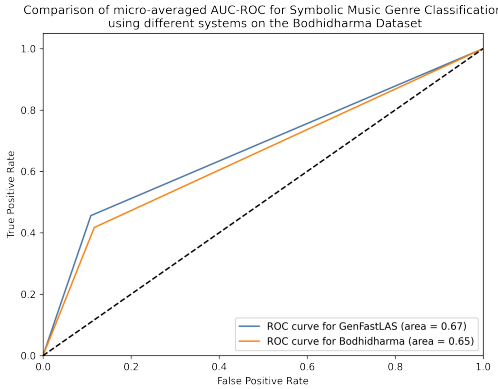
Figure 5.5: Comparison of AUC-ROC curves for the three systems trained on 150 datapoints of the Lakh MIDI Dataset. The bottom-right figure shows the comparison of micro-averaged AUC-ROCs.



(a) GenFastLAS



(b) Bodhidharma



(c) Comparison

Figure 5.6: Comparison of AUC-ROC curves for the two systems trained on the Bodhidharma Dataset. The bottom figure shows the comparison of micro-averaged AUC-ROCs.

GenFastLAS is now performing slightly better than both baselines, given its higher micro-averaged F1 score. This showcases the data efficiency with which GenFastLAS is able to approach our classification task. As a matter of fact, neither the Bodhidharma system nor MusicBERT are able to learn correctly from only 150 datapoints, and they require many more (around 142,697 more training points for both systems) in order to perform correctly.

This observation answers question (4), by showing that accuracy is lost when using GenFastLAS over other methods, but a high data efficiency is gained.

Using another dataset: The Bodhidharma Dataset

Finally, we evaluate both our GenFastLAS system and the Bodhidharma system on the *Bodhidharma Dataset* [12]. This dataset was specifically designed to be used within the Bodhidharma system, and contains an ensemble of 950 MIDI files, all labelled with one of 38 genre labels. These 38 genre labels are further defined as all belonging to one of 9 larger genres. We use the labels of

Model	F1 (micro-averaged)	Execution time (in seconds)
GenFastLAS	0.4561	8,395
Bodhidharma	0.4173	9

Table 5.3: Comparison of micro-averaged F1 scores and execution time for GenFastLAS and the Bodhidharma systems, when applied to the Bodhidharma Dataset.

this dataset and perform a direct mapping to 4 of our 6 genre labels that we have considered until now. Namely, the labels *Country* and *Jazz* are directly mapped to our genre labels of the same name; the labels *Modern Pop* and *Rock* are combined and mapped to our *Pop/Rock* genre label; and the subgenre *Latin* is mapped to our genre label of the same name. We therefore use 4 genre labels in this case.

Figure 5.6 shows the AUC-ROC curves obtained for both GenFastLAS and the Bodhidharma system when applied to the Bodhidharma Dataset. Table 5.3 presents the execution time and accuracy of each system in this context. We can observe that GenFastLAS is performing better than the Bodhidharma system on its own dataset, even when only trained on 150 datapoints. This suggests that the additional features that we developed bring important improvement.

5.2 Evaluating GenFastLAS in isolation

5.2.1 Comparing with another Symbolic learning task

In order to evaluate the improvement brought by GenFastLAS on a real-world FastLAS task, we look at the Online Learning of Anomaly detection Policies from Historical data (OLAPH) framework by Drozdov et al. [29]. OLAPH is a powerful anomaly detection framework that uses FastLAS in order to detect anomalies in an online fashion. Although OLAPH performs very well on its original dataset, it suffers from the standard scalability issues of FastLAS when increasing the number of features, as we previously discussed. We therefore evaluate the improvement that GenFastLAS brings when being integrated within OLAPH.

Splice-junction Gene Sequences Dataset

In order to evaluate our integration on a new dataset, we extract a one-class classification dataset from the Molecular Biology (Splice-junction Gene Sequences) Dataset from UCI's Machine Learning Repository [30]. The goal of this dataset is to detect boundaries between exons and introns within sequences of DNA. To transform it into an anomaly detection-like dataset, we extract only the negative class (`N`) and attempt to model the situation where a sequence of DNA has no exon/intron or intron/exon boundary. We therefore consider the presence of an exon/intron (`EI`) or an intron/exon (`IE`) boundary as an anomaly. We show here an excerpt from the dataset:

Class	DNA Sequence
`N`	`TACAATGCTCATAACATAAAAAATCAAACATAACATCAAATAAAATATATGATTCTGATA`
`EI`	`CCCTCGTGCGGTCCACGACCAAGACCAGCGGTGAGCCACGGGCAGGCCGGGGTCGTGGGG`
`IE`	`GGGCGGGTCTCAACCCCTCCTCGCCCCCAGGCTCCCACTCCATGAGGTATTTCAGCGCCG`

Since each DNA sequence is 60 nucleotide long, this allows us to work with a dataset with 60 categorical attributes. Each categorical attribute can take one of eight values, with four values being standard nucleotides (`A`, `G`, `T` and `C`) and four values encoding ambiguity, as explained in the dataset specifications.

We select 30 of these features and compare how Vanilla-OLAPH and GenFastLAS-OLAPH perform on the 1,655 datapoints with class `N`. Vanilla-OLAPH, due to its implementation, will run FastLAS tasks by considering every body mode declaration, while GenFastLAS will vary the set of body mode declarations as usual.

Performance comparison

The main improvement brought by GenFastLAS in the execution of OLAPH is the speed with which it is able to learn new policies from the provided historical data. Since Vanilla-OLAPH uses standard FastLAS tasks for each new policy, and given the high number of features in the dataset, it takes much longer to learn each policy. Table 5.4 shows the execution time and the example coverage for the first two policies learnt by both Vanilla-OLAPH (OLAPH) and GenFastLAS-OLAPH (G-OLAPH).

Overall, and over the course of the 24 hour experiment we ran, Vanilla-OLAPH was able to learn a total of 4 policies, while GenFastLAS-OLAPH learnt 491 policies, before running out of data after 3 hours and 41 minutes. This progression is shown on Figure 5.7.

Policy learnt	Number of examples	Execution time		Examples covered	
		OLAPH	G-OLAPH	OLAPH	G-OLAPH
1st Policy	10	5m 10s	**1m 21s**	100%	100%
2nd Policy	40	4h 6m 23s	**1m 11s**	47.5%	**100%**

Table 5.4: Comparison of the execution time and the example coverage for Vanilla-OLAPH (OLAPH) and GenFastLAS-OLAPH (G-OLAPH).

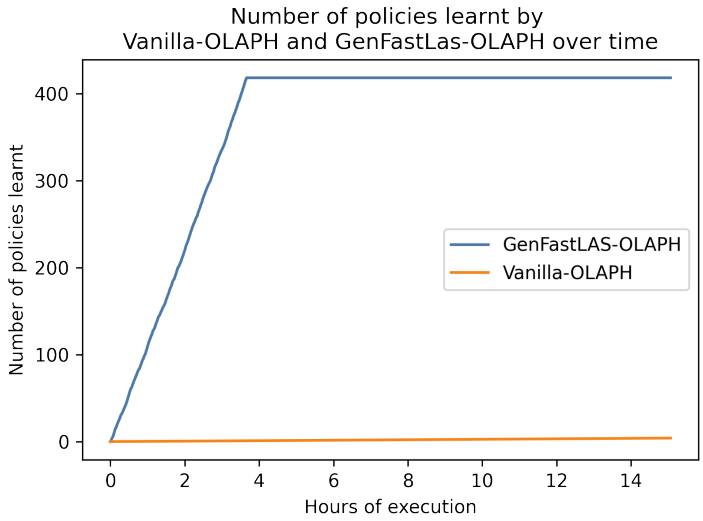


Figure 5.7: Comparison of the number of policies learnt by both Vanilla-OLAPH and GenFastLAS-OLAPH over time

A potential drawback of using GenFastLAS for OLAPH would be that, through the use of Genetic Algorithms, GenFastLAS might favour the best features and disregard the others, limiting the usefulness of the dataset. In order to verify whether that is the case, we compute the *Shannon Entropy* of the set of all the features used throughout every policy learnt, for both Vanilla-OLAPH and GenFastLAS-OLAPH. We then normalise this Entropy to compute the *Balance* of the features used throughout the policies learnt, which we define as follows.

Definition 20. *We define as Balance the normalised* Shannon Entropy*, which we express as*

$$Balance = \frac{H}{\log N} = \frac{-\sum_{i=1}^{N} \frac{f_i}{F} \log \frac{f_i}{F}}{\log N}$$

with H the Shannon Entropy*, N the number of features, f_i the number of times the feature with index i is used, and F the total number of times any feature is used.*

We compute this Balance for both methods and obtain the results shown on Table 5.5. We can see that, although GenFastLAS-OLAPH has a slightly lower *Balance* score than Vanilla-OLAPH, it remains very close to 100%, which indicates that the features are used in a fairly balanced way. This, in turn, means that GenFastLAS is able to explore the feature space efficiently in this application. We can explain GenFastLAS's slightly less balanced feature usage by its tendency to stick to the best features once they are computed. We note that this behaviour can be modulated through the random initialisation hyperparameter.

	OLAPH	G-OLAPH
Balance	**99.11%**	95.84%

Table 5.5: Comparison of the Balance computed for both Vanilla-OLAPH (OLAPH) and GenFastLAS-OLAPH (G-OLAPH).

	OLAPH	G-OLAPH
Final Accuracy	1.15%	**11.48%**

Table 5.6: Comparison of the final accuracy computed for both Vanilla-OLAPH (OLAPH) and GenFastLAS-OLAPH (G-OLAPH), for the last policy learnt by both in the scope of 24 hours.

Feature	mRMR Rank
Chord Progression	1/105
Sections	6/105
Hi-Hat Pattern	10/105
Snare Pattern	15/105
Kick Pattern	23/105
Cymbal Pattern	44/105
Other Pattern	46/105
Toms Pattern	60/105

Table 5.7: mRMR rank for each of our features. In our example, 105 features are used.

Finally, we evaluate the accuracy of each final policy learnt using the *OPA* tool as suggested by Drozdov et al., and list the results on Table 5.6. The very low accuracies are explained by the very challenging nature of the dataset we picked. We note that, ideally, we would have evaluated both methods on the original dataset, but it was not available for privacy reasons. However, it can still be noted that GenFastLAS-OLAPH greatly surpasses Vanilla-OLAPH in terms of final accuracy. This is very probably explained by the fact that GenFastLAS-OLAPH was able to traverse the entire dataset during the experiment, whereas G-OLAPH only iterated through a small part of it. However, this proves that GenFastLAS is able to correctly transform a system in order to improve both its speed and accuracy, answering questions (1) and (2). It is also to be noted that Vanilla-OLAPH reaches an especially low accuracy, even though it has access to all of the 30 features, whereas GenFastLAS-OLAPH reaches a higher accuracy by only attempting different combinations of those features.

5.3 Evaluating our new symbolic music features

We now focus on evaluating the new symbolic music features that we described in Section 4.2. We first start by evaluating their usefulness and relevance within our Symbolic Music Genre Classification task over the Lakh MIDI Dataset, before evaluating their explainability.

5.3.1 Evaluating relevance

In order to evaluate the relevance of the features we developed within our task of Symbolic Music Genre Classification, we first start by observing how frequently our features are picked by GenFastLAS when exploring the hypothesis space. Since GenFastLAS selects features initially using the mRMR metric (see Subsection 3.2.1), we can observe how frequently our features get selected given different initial number of features that GenFastLAS considers. To do so, we consider the entire Lakh MIDI Dataset that we process as previously. Table 5.7 shows the rank of each of our features in the initial mRMR ranking. We can observe that the best ranked feature is the Chord Progression feature, and that, generally, our features are almost all ranked in the first half of this ranking.

We can also observe how frequently our features are used within the solutions learnt by GenFastLAS. Figure 5.8 shows the evolution of the average proportion of our features and jSymbolic features within solutions, given different levels of fitness computed for each solution.

Together with the evaluation from Subsection 5.1.2 when using the Bodhidharma Dataset, we conclude that our features are highly relevant and generally improve the fitness and accuracy of our solutions. We now attempt to evaluate their explainability.

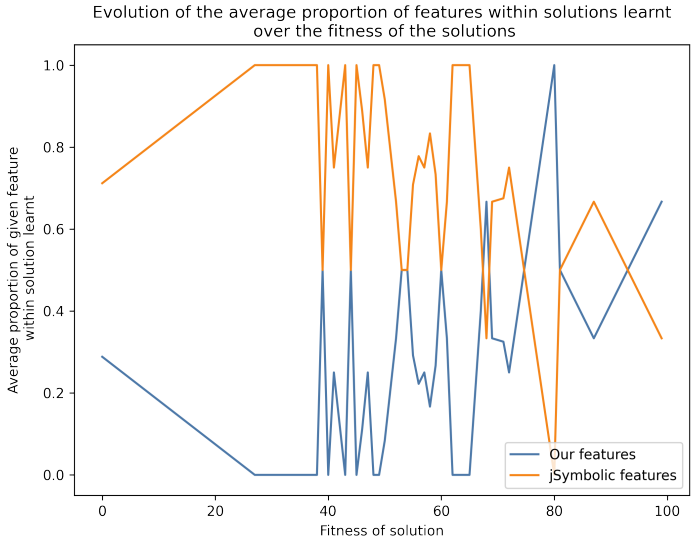


Figure 5.8: Evaluation of the proportion of our features within the solutions learnt over the fitness of those solutions.

5.3.2 Evaluating explainability

An important benefit brought by the use of our symbolic music features is the explainability that is associated with them, due to the human-infused music theory knowledge that they are refer to. As mentionned in Chapter 2, explainability is an essential characteristic of human-centric AI, and one that we aimed to put forward in our system. It generally allows more interactivity in the use of AI, and, in our case, offers learning opportunities out of the solutions learnt by our system.

However, evaluating explainability is hard, and there is no current consensus on any framework or metric that can consistently measure the explainability of a set of features, or of a system. Generally, explainability is evaluated through user testing and surveys, and is measured as the extent to which human users are able to "understand" the solutions learnt by a system. In that line of thought, we decided to develop our own survey to put our features to the test.

Developing an explainability survey

In order to evaluate the explainability of our features, and, incidently, of our classification algorithm, we developed a user-facing survey to measure the extent to which human users were able to understand the solutions learnt by different systems. The survey, due to its esoterical content, was directed towards a panel of various people, who all stated they had a musical education background. This is in order to evaluate purely the explainability of the answers, and not the ability of the algorithms to simplify musical background knowledge, which would be a completely different task.

We therefore create a survey to measure how easily users can understand the solutions learnt by GenFastLAS, and using our features, compared to the solutions learnt by the Bodhidharma system, using the jSymbolic features. Figure A.1, in Appendix, shows the four questions that were used as part of our explainability survey. Users were given a part of the solutions learnt by the different systems, and had to guess the genre that this part of the solution encoded for. The solutions that were used for each questions were extracted following different processes:

- For the two first questions that use our features and our GenFastLAS system, we extracted random rules that included our features out of the solution with the best accuracy over all folds, and filtered out the jSymbolic features in every rule. We then designed a visual representation to present this solution.
- For the other two questions that use the jSymbolic features and the Bodhidharma system, we had to manually process the system in order to first inject some explainability. As a matter of fact, interpreting the high-dimensional result of a SVM is difficult, and we needed

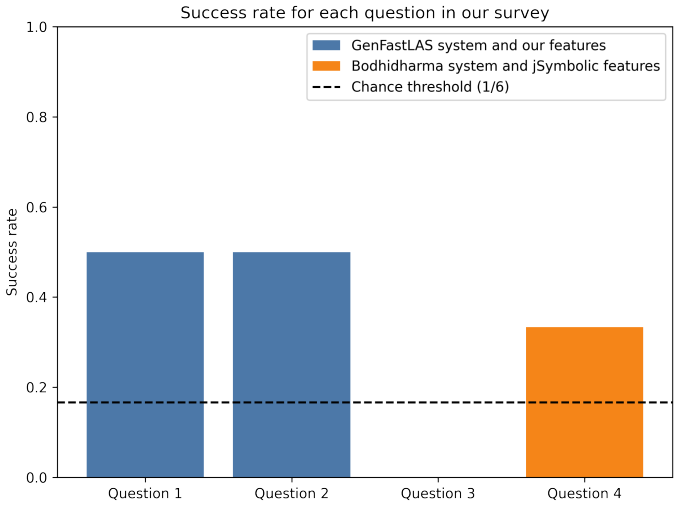


Figure 5.9: Success rate for each question in our survey.

to only consider a part of it to extract a meaningful solution. We first started by running the Bodhidharma system over a set of 2 randomly chosen genre labels, in order to reduce the space to a two-dimensional space. We then extracted the two most representative features out of the SVM, and reran the system using only those two features. We then extracted the 2D support vector and represented it as shown on the Figure. We also added a description of the two jSymbolic features used.

Extracting an explainability metric

Over the course of one week, during which this survey has been distributed and monitored, 6 responses were received. We plot the success rate of each question on Figure 5.9. We can notice that the two questions that included solutions learnt by our GenFastLAS system, and using our features, obtained a much higher success rate than the two others that included solutions learnt by the Bodhidharma system, and using the jSymbolic features. Additionally, both questions using GenFastLAS were above the chance line, whereas only one of the two using Bodhidherma was. This chance line is set at 1/6 since six possible answers were provided for each question. As such, we argue that our solutions tend to be much more understandable by humans, and this therefore answers question (7).

Chapter 6

Related Work

This chapter reviews previous work related to the areas that this thesis touches upon. We first start by studying different technologies that integrate Logic-based Learning with Genetic Algorithms, before exploring previous work on the extraction of features out of symbolic music data.

6.1 Logic-based Learning and Genetic Algorithms

Previous work has been carried out in the past thirty years that aims to apply Genetic Algorithms to perform logic-based learning.

Giordana & Sale [31] introduce GA-SMART, which uses Genetic Algorithms to induce concept descriptions in conjunctive normal form. Giordana & Saitta [32] extend on this work, and propose REGAL, which enables the induction of concept descriptions expressed in the language of First-Order Logic.

Venturini [33] introduces a supervised genetic learning system called SIA, which uses attribute/value representations, and applies genetic processes in order to generalise sampled examples and learn rules that maximise a noise-tolerant rule-evaluation criterion. Augier et al. [34] further introduce SIAO1, which extends the SIA system to cover First-Order Logic representations.

Other kinds of integration between those two classes of algorithms have also been explored. Gandhi [35] approaches the problem in another direction, and uses Inductive Logic Programming to attempt and explain the solutions generated by a genetic algorithm performing a scheduling operation on job shop problems. Milaré et al. [36], similarly, explore the integration of Genetic Algorithms with Symbolic Learning systems to extract rules from Artificial Neural Networks (ANNs), by using Genetic Algorithms to mutate rules learnt by Symbolic Learning systems and evaluating their infidelity rate towards the ANN solutions.

However, no work, to the best of our knowledge, has been carried out to use recent genetic algorithm strategies to optimise some aspects of the existing Logic-based Learning systems, such as what GenFastLAS proposes.

6.2 Symbolic music features

We now explore the different features that have been developed to perform Music Information Retrieval tasks on symbolic music data, as well as the different MIR tasks that they have been applied to.

McKay [22], as mentioned previously, proposes a catalogue of 111 features that can be extracted from MIDI files, and that all aim to target one of seven different musical categories: Instrumentation, Texture, Rhythm, Dynamics, Pitch Statistics, Melody, and Chords. When applied on the Bodhidharma Dataset with a ensemble of 10 genres, an accuracy of 66.4% is reached.

Kotsifakos et al. [37] propose a model which, taking MIDI as an input, transforms all channels into sequences of 2D points which are then used as a query to a kNN classifier. The model also uses a similarity measure called Subsequence Matching with Bounded Gaps and Tolerances (SMBGT) to determine the similarity between songs and create a similarity matrix used inside the kNN classifier. The model was applied to a corpus of 100 MIDI files freely available online that cover four genres of music (classical, blues, rock and pop) and achieves an accuracy of 40%.

Valverde-Rebaza et al. [38] extract features from MIDI representations using histograms of notes and statistical moments. Classification is then applied using four traditional classification approaches: decision trees, naive Bayes, multilayer perception with backpropagation, and support vector machines. Classification is also performed through four relational classifiers: probabilistic relational neighbour, weighted vote relational neighbour, network-only Bayes, and network-only link-based, that use three variations of the kNN strategy, symmetric kNN, mutual-kNN, and regular kNN. The model was applied on a music collection which includes 919 MIDI files manually classified into four genres: classical, brazilian backcountry, pop/rock, and jazz. The best performance is achieved through a network-only link-based classifier using class-distribution-link in a regular kNN network and using musical structure as a feature input. The achieved AUC-ROC score is .967.

Previous work has also been done on classifying music by using both audio and symbolic features. Lidy et al. [39] use an automatic music transcription system to extract symbolic descriptors from a raw audio file, which are then passed as an input together with the audio file itself to a Support Vector Machine classifier. The transcription system converts the audio signal into a MIDI file that only considers pitches and note durations. This conversion is performed through STFT, followed by onset detection through peak detection, and pitch detection. 37 symbolic features are then extracted from the transcribed notes. The features extracted from the audio are rhythm patterns, rhythm histograms, statistical spectrum descriptors, as well as onset features. Using both the symbolic and audio features in the classification, an accuracy of 76.8% on the GTZAN dataset is achieved.

More recently, Zeng et al. [28] have proposed MusicBERT, a pre-trained model based on a Transformer encoder architecture, to use MIDI input data to perform various MIR tasks, including melody completion, accompaniment suggestion and genre classification. A novel encoding, called OctupleMIDI, is also introduced to interact with the BERT architecture, and represents each note inside the MIDI file as a tuple of eight elements. A bar-level masking strategy is also developed as a better masking strategy to use within the domain of music, inside the pre-training mechanism for MusicBERT. When applied to a newly collected dataset of more than one million MIDI files, MusicBERT reaches a F1 score of up to 0.784 for the genre classification task.

Chapter 7

Conclusion and Future Work

7.1 Summary of contributions

Throughout this project, we have introduced *GenFastLAS*, a scalable Logic-based Learning system that can learn explainable solutions to Machine Learning problems, and scale up with respect to the complexity of the data it is applied to. Our algorithm integrates the MAP-Elites QD algorithm with the FastLAS learning system, and allows the former to efficiently direct the exploration of the hypothesis space of the latter. In order to make this integration possible, we introduced a novel genetic encoding of Learning from Answer Sets tasks. We also discussed different behaviour descriptors that can be used to better guide the exploration of solutions, and introduced the idea of making the learning of GenFastLAS more interactive through an online selection of behaviour descriptors.

We have also discussed the requirement of having explainable features in order to learn explainable solutions that can be used within the areas of research, creativity and education. We have then reviewed the current availability of explainable features for symbolic music data, and explored methods to develop new ones. We have introduced a set of eight new features which are all informed with music theory knowledge, and which can be applied to a variety of Music Information Retrieval tasks.

Finally, we have evaluated both GenFastLAS and our new features, in integration and in isolation, and compared them with existing systems. For increasing numbers of data features, GenFastLAS consistently offers a gain of speed of several orders of magnitude when compared to FastLAS, without suffering from a decrease in accuracy. In some cases, GenFastLAS is capable of computing solutions for which FastLAS times out after more than twenty-four hours. When applied to increasing numbers of datapoints, GenFastLAS, while grappling with scaling up in this direction, still executes much faster than FastLAS, without almost any decrease in accuracy. When applied to the task of Symbolic Music Genre Classification, GenFastLAS is unable to correctly compete with other systems, due to its unability to manage such large amounts of data. However, GenFastLAS displays a high data efficiency, which is demonstrated by its ability to overperform the other systems when applied to the same small amount of data. In other areas of application, GenFastLAS is also able to greatly improve the performance of systems which traditionally use FastLAS for their learning. We have also evaluated our new symbolic music features, which proved to be highly relevant to the task of Symbolic Music Genre Classification, and offered a higher level of explainability than existing features.

7.2 Future Work

While GenFastLAS offers a novel integration between Logic-based Learning systems and Genetic Algorithms ,and enables the scalable learning of explainable solutions for complex data, we believe that the range of opportunities opened up by such an integration has only been partially explored. We aim to mention a few of those opportunities that, we believe, can help push the state of research in Logic-based Learning to new horizons.

- As previously shown, GenFastLAS is much more able to scale up with respect to the number of features than to the amount of data in the training set. We believe that the genetic

encoding of the FastLAS tasks can be enhanced to additionally embed a specific selection of examples, which would allow GenFastLAS to also scale up in this direction.

- We have mentioned that behaviour descriptors offer a promising opportunity for interactive applications of the system, and we believe that defining such descriptors and taking inspiration from interactive applications of MAP-Elites can be of great value to the areas of research, creativity, and education.
- Following our definition of explainable features and our discussion of their necessity, we believe that there is some very interesting line of work for future automated feature extraction, which could happen by automatically collecting human knowledge from online encyclopediae. By enabling the processing of large amounts of features, GenFastLAS could then be considered as a learning system of choice for the explainable processing of those new features.
- Finally, we believe that additional work in integrating any type of optimisation algorithm with Logic-based Learning systems can be incredibly beneficial for the improvement of those systems, and, incidently, for the improvement of the state of research of Explainable AI and Human-AI Interaction.

Chapter 8

Ethical Discussion

Given its potential for generalisation, GenFastLAS, like many other AI systems, embeds important ethical considerations that need to be acknowledged before any application. Nevertheless, we believe that the particular area that we explored in this thesis, namely Music Information Retrieval, contains little ethical considerations. Additionally, due to the opportunities for Human-AI interactivity offered by GenFastLAS, considerations about the scope and legitimacy of this interactivity need to be examined before any application.

Since GenFastLAS makes use of two open-source algorithms, there is no added legal implication in the use and sharing of this project. In the event of GenFastLAS using, or being used, in a commercial context, some form of licensing will be required.

Finally, it is also to be noted that the music datasets used, together with the music theory that we have referred to, focus almost exclusively on music produced within the Western world over the last five centuries. As such, the observations and conclusions that we have drawn do not intend, in any way, to reflect the real state and variety of music on a global scale.

Appendix A

Appendix

Sections:
A - B - C - B - C - A

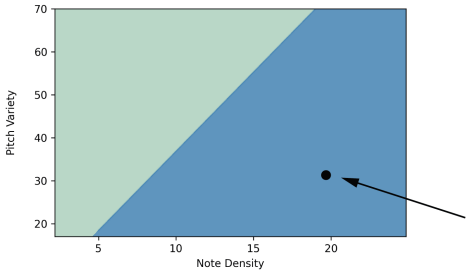
Chord Progression:
II - V - I

(a) Question 1. Answer: Jazz

Sections:
A - B - A - B - C - B

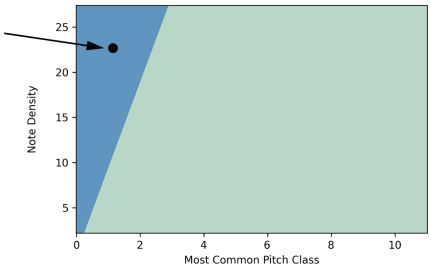
Chord Progression:
VI - IV - V

(b) Question 2. Answer: Pop/Rock

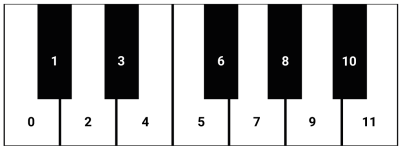


with
Pitch Variety: Number of pitches used at least once
Note Density: Average number of notes per second

(c) Question 3. Answer: Jazz



with
Note Density: Average number of notes per second
Most Common Pitch Class: Label of the most common pitch, annotated as such:



(d) Question 4. Answer: Pop/Rock

Figure A.1: The four diagrams used in our explainability survey. The two diagrams at the top are extracted from the results of GenFastLAS, while the two diagrams at the bottom are extracted from the results of the SVM of the Bodhidharma system.

Bibliography

[1] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. [Preprint], April 2015. URL http://arxiv.org/abs/1504.04909.

[2] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using Centroidal Voronoi Tessellations to Scale Up the Multi-dimensional Archive of Phenotypic Elites Algorithm. [Preprint], July 2017. URL http://arxiv.org/abs/1610.05729.

[3] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. FastLAS: Scalable Inductive Logic Programming Incorporating Domain-Specific Optimisation Criteria. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2877–2885, April 2020. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v34i03.5678. URL https://mx.aaai.org/ojs/index.php/AAAI/article/view/5678.

[4] Kristen Stubbs, Pamela J. Hinds, and David Wettergreen. Autonomy and Common Ground in Human-Robot Interaction: A Field Study. *IEEE Intelligent Systems*, 22(2):42–50, March 2007. ISSN 1941-1294. doi: 10.1109/MIS.2007.21. Conference Name: IEEE Intelligent Systems.

[5] Tim Miller. Explanation in Artificial Intelligence: Insights from the Social Sciences. [Preprint], August 2018. URL http://arxiv.org/abs/1706.07269.

[6] Herbert H. Clark and Susan E. Brennan. *Perspectives on socially shared cognition*. American Psychological Association, Washington, DC, 4. printing edition, 1991. ISBN 978-1-55798-376-3.

[7] Mark Law, Alessandra Russo, and Krysia Broda. Inductive Learning of Answer Set Programs. In *Logics in Artificial Intelligence*, volume 8761, pages 311–325. Springer International Publishing, Cham, 2014. ISBN 978-3-319-11557-3 978-3-319-11558-0. URL http://link.springer.com/10.1007/978-3-319-11558-0_22.

[8] Mark Law, Alessandra Russo, and Krysia Broda. Simpli[U+FB01]ed Reduct for Choice Rules in ASP. Technical Report DTR2015-2, Imperial College of Science, Technology and Medicine, Department of Computing, April 2015.

[9] MIDI Manufacturers Association. MIDI Manufacturers Association Website, 2022. URL https://www.midi.org/. Accessed 2022-06-19.

[10] Michael Scott Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, pages 637–642. International Society for Music Information Retrieval, 2010.

[11] Colin Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, 2016.

[12] Cory McKay. Automatic Genre Classification of MIDI Recordings. Master's thesis, Department of Music Research, Schulich School of Music, McGill University, 2004.

[13] INRIA. Python3 MAP-Elites Github Repository, 2022. URL https://github.com/resibots/pymap_elites. Accessed 2022-06-19.

[14] Chris Ding and Hanchuan Peng. Minimum redundancy feature selection from microarray gene expression data. *Journal of Bioinformatics and Computational Biology*, 3(2):185–205, April 2005. ISSN 0219-7200. doi: 10.1142/s0219720005001004.

[15] Zhenyu Zhao, Radhika Anand, and Mallory Wang. Maximum Relevance and Minimum Redundancy Feature Selection Methods for a Marketing Machine Learning Platform. [Preprint], August 2019. URL http://arxiv.org/abs/1908.05376.

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. [Preprint], June 2018. URL http://arxiv.org/abs/1201.0490.

[17] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor-a hunter of idle workstations. In *Proceedings. The 8th International Conference on Distributed*, pages 104–111, June 1988. doi: 10.1109/DCS.1988.12507.

[18] Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1950.tb00463.x.

[19] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, November 1953. ISSN 2379-674X. doi: 10.1109/TCE.1953.6371932.

[20] Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. Interactive constrained map-elites: Analysis and evaluation of the expressiveness of the feature dimensions. *IEEE Transactions on Games*, 14(2):202–211, 2022. doi: 10.1109/TG.2020.3046133.

[21] Alexander Schindler, Rudolf Mayer, and Andreas Rauber. Facilitating Comprehensive Benchmarking Experiments on the Million Song Dataset. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR 2012)*. International Society for Music Information Retrieval, 2012.

[22] Cory McKay. *Automatic Music Classification with jMIR*. PhD thesis, Department of Music Research, Schulich School of Music, McGill University, 2010.

[23] Bruce Benward. *Music in theory and practice*. W.C. Brown, Dubuque, Iowa, 3rd ed edition, 1985. ISBN 978-0-697-03621-6 978-0-697-03633-9.

[24] Masataka Goto. A chorus section detection method for musical audio signals and its application to a music listening station. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1783–1794, September 2006. ISSN 1558-7924. doi: 10.1109/TSA.2005.863204. Conference Name: IEEE Transactions on Audio, Speech, and Language Processing.

[25] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, Baltimore, 3rd ed edition, 1996. ISBN 978-0-8018-5413-2 978-0-8018-5414-9.

[26] Genius.com. Lyrics for Blur – Charmless Man, 2022. URL https://genius.com/Blur-charmless-man-lyrics. Accessed 2022-06-12.

[27] MIDI Manufacturers Association. General MIDI 1 Sound Set, 2022. URL https://www.midi.org/specifications-old/item/gm-level-1-sound-set. Accessed 2022-06-14.

[28] Mingliang Zeng, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, and Tie-Yan Liu. MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 791–800, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.70. URL https://aclanthology.org/2021.findings-acl.70.

[29] Arthur Drozdov, Mark Law, Jorge Lobo, Alessandra Russo, and Mercion Wilathgamuwage Don. Online Symbolic Learning of Policies for Explainable Security. In *Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 269–278, December 2021. doi: 10.1109/TPSISA52974.2021.00030.

[30] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. URL http://archive.ics.uci.edu/ml. Accessed 2022-06-05.

[31] Attilio Giordana and Claudio Sale. Learning structured concepts using genetic algorithms. In *Machine Learning Proceedings 1992*, pages 169–178. Morgan Kaufmann, San Francisco (CA), 1992. ISBN 978-1-55860-247-2. doi: https://doi.org/10.1016/B978-1-55860-247-2.50027-9. URL https://www.sciencedirect.com/science/article/pii/B9781558602472500279.

[32] Attilio Giordana and Lorenza Saitta. REGAL: An Integrated System for Learning Relations Using Genetic Algorithms. In *Proceedings of the International Workshop on Multistrategy Learning*, volume 2nd, pages 234–249, Harpers Ferry, West Virginia, May 1993.

[33] Gilles Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. In *Machine Learning: ECML-93*, volume 667, pages 280–296, Berlin, Heidelberg, April 1993. Springer Berlin Heidelberg. ISBN 978-3-540-56602-1 978-3-540-47597-2. doi: 10.1007/3-540-56602-3_142. URL http://link.springer.com/10.1007/3-540-56602-3_142.

[34] S Augier, G Venturini, and Y Kodratoff. Learning First Order Logic Rules with a Genetic Algorithm. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, KDD'95, page 21–26. AAAI Press, 1995.

[35] Sachin Gandhi. *Learning from a Genetic Algorithm with Inductive Logic*. PhD thesis, Ohio University, June 2005.

[36] Claudia R. Milaré, Gustavo E. A. P. A. Batista, André Carlos P. L. F. de Carvalho, and Maria C. Monard. Applying Genetic and Symbolic Learning Algorithms to Extract Rules from Artificial Neural Networks. In *MICAI 2004: Advances in Artificial Intelligence*, volume 2972, pages 833–843. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-21459-5 978-3-540-24694-7. doi: 10.1007/978-3-540-24694-7_86. URL http://link.springer.com/10.1007/978-3-540-24694-7_86.

[37] Alexios Kotsifakos, Evangelos E. Kotsifakos, Panagiotis Papapetrou, and Vassilis Athitsos. Genre classification of symbolic music with SMBGT. In *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '13*, pages 1–7, Rhodes, Greece, 2013. ACM Press. ISBN 978-1-4503-1973-7. doi: 10.1145/2504335.2504382. URL http://dl.acm.org/citation.cfm?doid=2504335.2504382.

[38] Jorge Valverde-Rebaza, Aurea Soriano, Lilian Berton, Maria Cristina Ferreira de Oliveira, and Alneu de Andrade Lopes. Music Genre Classification Using Traditional and Relational Approaches. In *2014 Brazilian Conference on Intelligent Systems*, pages 259–264, Sao Paulo, Brazil, October 2014. IEEE. doi: 10.1109/BRACIS.2014.54. URL http://ieeexplore.ieee.org/document/6984840/.

[39] Thomas Lidy, Andreas Rauber, and Antonio Pertusa. Improving Genre Classification by Combination of Audio and Symbolic Descriptors using a Transcription System. In *Proceedings of the 8th International Society for Music Information Retrieval Conference (ISMIR 2007)*, pages 61–66. International Society for Music Information Retrieval, 2007.