OPTIMIZING FEATURE EXTRACTION FOR SYMBOLIC MUSIC

Federico Simonetta[1] **Ana Llorens**[2] **Martín Serrano**[1]
Eduardo García-Portugués[3] **Álvaro Torrente**[1,2]
[1] ICCMU - Instituto Complutense de Ciencias Musicales, Madrid
[2] Universidad Complutense de Madrid, Madrid
[3] Universidad Carlos III, Madrid
didone@iccmu.es

ABSTRACT

This paper presents a comprehensive investigation of existing feature extraction tools for symbolic music and contrasts their performance to determine the set of features that best characterizes the musical style of a given music score. In this regard, we propose a novel feature extraction tool, named musif, and evaluate its efficacy on various repertoires and file formats, including MIDI, MusicXML, and **kern. Musif approximates existing tools such as jSymbolic and music21 in terms of computational efficiency while attempting to enhance the usability for custom feature development. The proposed tool also enhances classification accuracy when combined with other sets of features. We demonstrate the contribution of each set of features and the computational resources they require. Our findings indicate that the optimal tool for feature extraction is a combination of the best features from each tool rather than those of a single one. To facilitate future research in music information retrieval, we release the source code of the tool and benchmarks.

1. INTRODUCTION

Feature extraction is a pivotal task in contemporary machine learning. Music features can be categorized into two main types: symbolic and audio. While audio features have been subject to extensive research, computational techniques for symbolic music remain comparatively underexplored.

In recent years, there has been an increasing interest in analyzing symbolic scores in music. This encompasses studies on composer [1] and style recognition [2], affective computing [3], music generation [4], analysis of performance [5], and interpretation [6]. The symbolic dimension of music concerns the conceptual representation of musical data [7]. This level has been used in the field of Music Information Retrieval (MIR), with particularly successful outcomes when employed to support multimodal approaches [8], which integrate both audio and symbolic levels through audio-to-score alignment techniques [9]. The symbolic level is also crucial for musicologists, as music scores are the most common source for historical music studies. Musicologists rely on computational tools to extract and analyze musical scores on a large scale [10, 11]. However, traditional manual annotations, such as harmony [12] and cadence [13], are time-consuming and prone to errors. Therefore, computational tools are essential for efficient and accurate musicological analysis. Presently, two primary tools are available for extracting features from symbolic music: jSymbolic [14] and music21 [15]. Although both tools are open-source and widely employed, no comprehensive comparison between them has been conducted yet.

In this paper, we propose a novel set of features that is specifically, although not exclusively, tailored for the analysis of 18th-century Italian opera. We have developed a tool for extracting these features, named musif, that is being used for the analysis of operatic music in the Didone project [1] [16]. Here, we conduct a comparative study between musif and other existing tools, thus providing valuable insights into the strengths and weaknesses of each of them. Additionally, we evaluate the efficiency of each tool and demonstrate that musif adds useful features to both music21 and jSymbolic. We observe that, in most cases, a combination of features from multiple tools yields the most powerful feature set. To validate our findings, we test all three tools on various repertoires. We aim to compare the feature sets on file formats with varying levels of representation abilities, such as MIDI, MusicXML, and **kern. While MIDI is widespread in computational studies, it is relatively simplistic for written music; MusicXML and **kern, instead, are less commonly utilized in MIR but provide more accurate representations when dealing with music scores.

The main contributions of this paper are, therefore, threefold. Firstly, we present a new set of features designed for the study of an under-represented repertoire in music computing literature, i.e., 18th-century Italian opera. Secondly, we introduce musif, a new efficient, extensible, and open-source Python tool for feature extraction from symbolic music. Finally, we provide a benchmark of music21,

[1] https://didone.eu

jSymbolic, and musif on a variety of repertoires and file formats.

The whole code used for this study, as well as the code used for the proposed tool, is available at `https://github.com/DIDONEproject/music_symbolic_features/`.

2. FEATURE EXTRACTION TOOLS

In this study, we compare three tools for feature extraction from symbolic music: jSymbolic [14], music21 [15], and musif. Other tools such as Humdrum [2] may be used for feature extraction, but they would require a larger effort for assembling different features from various toolkits and organizing them in a usable tabular format. We will describe each one in detail in the following subsections.

2.1 jSymbolic

The jSymbolic tool was initially introduced in 2006 [17] and subsequently updated in 2018 [14]. It is an open-source, Java-based software designed to extract features from both MIDI and MEI files. The latest iteration of jSymbolic is capable of extracting 246 distinct features, some of which are multidimensional and account for a total of 1022 values. However, the actual number of extracted features may vary depending on the user's configuration and the musical composition itself. jSymbolic features relate to pitch statistics, melodic intervals, chords and vertical intervals, rhythm, instrumentation, texture, and dynamics. In addition to these features, jSymbolic is capable of computing certain characteristics that are not readily available in MIDI files. To achieve this, jSymbolic utilizes the MEI file format to determine the number of slurs and grace notes in a given piece. While MEI and other high-informative file formats offer additional features such as pitch names, harmonic analysis, and written dynamic or agogic indications, jSymbolic does not take these into consideration.

The jSymbolic software provides users with the flexibility to customize configurations and features, facilitating the integration of previously existing feature values into newer features. Furthermore, users can extract windowed features by specifying window size and overlap in seconds. jSymbolic does not provide pre-built methods for parallel processing of large corpora, thereby requiring the user to implement a suitable strategy. Lastly, jSymbolic provides output options in both CSV and Weka's ARFF format.

The software is accessible as a self-contained program featuring a Graphical User Interface (GUI) and a Command Line Interface (CLI), as well as as a Java library.

2.2 music21

music21 is a Python toolkit designed for computational music analysis, which was first introduced in 2010 [18]. One of its remarkable features is the capability to parse a wide range of file formats, including MIDI, MusicXML, **kern, ABC, and various others. The music information is represented in an object-oriented hierarchical structure that is aimed at facilitating the development of novel tools.

After its initial academic publication, music21 was further developed with a set of features presented in 2011 [15]. The latest version of music21 includes 69 features introduced by jSymbolic, as well as 20 characteristics computed using the information parsed from high-informative file formats. These characteristics are related to key, cadence, harmony, and lyrics. Regardless of the input file format, music21 consistently outputs 633 features. However, the number of extracted features may vary since some features are zeroed out when they are not computable.

music21 is a Python module that lacks a CLI or a GUI. It does not have a configuration format; rather, it offers a broad range of methods for developing custom pipelines for different types of music information processing. These methods encompass the creation of new features and some automated high-level inference of music characteristics, such as key [19], as well as tools for windowed analysis.

One disadvantage of music21 is that large music scores may result in deeply nested Python objects with numerous non-picklable attributes attached. This makes the programming process challenging, particularly due to the difficulty of saving these objects to a file.

In this study, we have developed a CLI for utilizing music21 feature extraction tools in a manner comparable to musif. This implementation facilitates parallel processing by distributing the extraction of features across numerous files simultaneously.

2.3 musif

Our software is named musif [20]. It is implemented in Python and built upon the music21 library, and offers an Application Programming Interface (API) with no default settings of significance and a CLI with default settings optimized for most common use cases.

We leverage music21's internal representation, enabling us to extract features from any file format supported by music21. musif is highly customizable and allows users to add custom features as required. After creating the internal representation of the musical score using music21, we extract multiple features and store them in `pandas` dataframes. This facilitates exporting results in various formats, making musif easily integrable into diverse pipelines.

One limitation of music21 is its restricted ability to serialize complex and large music scores. This restriction also affects the possibility of parallel processing, as Python's single-thread approach necessitates parallelization via processes, which in turn requires context copying and data serialization. Furthermore, parsing large XML files is one of the slowest steps in the feature extraction process. To optimize this procedure, a more favorable strategy would be to store the parsed XML files' logical structure on disk as a cache. We have thus implemented a caching system capable of caching and serializing any music21 object. A restriction to note about the caching system is that the cached scores are read-only. However, this feature enables the writing of parsed scores onto disk and caching of the output from resource-intensive music21 functions into memory.

musif can extract harmony-related features by utilizing standardized harmonic analyses annotated in the Mus-

[2] `https://github.com/humdrum-tools/humdrum-tools`

Table 1. Computational efficiency of the three feature extraction tools. Each run was repeated twice and the second run times are indicated between parentheses.

File format	Tool	Avg CPU Time (s)	Avg Real Time (s)	Avg RAM (GB)	Max RAM (GB)	Tot. errored files	Tot. files
MIDI	*musif*	66.30 (13.30)	5.62 (**1.14**)	9.10 (10.1)	14.2 (19.6)	1	16734
	music21	55.3 (55.2)	4.72 (4.71)	**7.12 (7.12)**	**9.87** (**9.94**)	**0**	
	jSymbolic	**2.20 (2.20)**	**1.98** (1.97)	7.97 (7.14)	16.1 (11.7)	14	
MusicXML	*musif*	15.4 (**6.63**)	1.32 (**0.57**)	5.87 (5.12)	12 (10.1)	4	14712
	music21	**10.8** (10.8)	**0.91** (0.91)	**4.30 (4.33)**	**5.68 (5.55)**	**0**	
***kern*	*musif*	26 (**13.1**)	2.26 (**1.14**)	5.20 (4.14)	5.60 (4.92)	**0**	472
	music21	**14.0** (14.1)	**1.21** (1.21)	**3.08 (3.04)**	**4.12 (4.18)**	**0**	

eScore file format [12, 13]. Besides, it encompasses a wide range of features, including melodic intervals, harmony, dynamics, tempo, density and texture, lyrics, instrumentation, scoring, and key. Notably, dynamics and tempo are determined by the composer's text notation rather than by MIDI parameters. Furthermore, our implementation includes all features provided by music21 with the exception of 14 features that utilized the caching system in writing mode. The number of extracted features depends on the complexity of the score and is influenced by both the number of parts and musif's compatibility with the encoding.

NaN values are used to represent non-computable features in a score. For example, when processing datasets with varying instrumentations, some features may not be available for all scores. These values can be replaced with a default value (e.g., 0) or removed from the corpus by deleting either the score or the related feature. In the CLI, we have implemented a heuristic to determine whether a score should be removed from the extracted corpus if it contains too many NaNs. Specifically, we define r as the ratio between the number of columns without NaN and the total number of rows in the output table. If $r < 0.1$, we compute n_i, which is the number of NaNs in the ith row. We remove rows with n_i greater than $\frac{1}{0.99}q_{0.99}$, where $q_{0.99}$ is the 99% quantile of $\{n_1, n_2, \ldots\}$, indicating that 99% of rows are not deleted. The factor $\frac{1}{0.99}$ can be better understood as dividing the $Q_{0.99}$ by 99, thus obtaining an estimate of $Q_{0.01}$, and multiplying it by 100, thus obtaining the expected value of $Q_{1.00}$ based on the first 99% of the data. Put differently, it computes the maximum n_i that we expect if the remaining 1% of rows has a number of NaN "similar" to the previous 99%. Larger values are thus considered outliers. This method was empirically tested on the corpuses used in this work (see Section 3), revealing that only a few scores were generally removed while most lines of the output table were retained. In case a score is not deleted, the CLI removes from the tale the features that are NaN in that score.

musif also incorporates a post-processing module that facilitates the removal, merging, or substitution of values in specific columns or groups of columns within the extracted data. This functionality proves especially advantageous when dealing with large tables generated by musif from a substantial set of scores, as it minimizes the computational effort required for processing such tables.

Like the other tools, we have implemented the capability to extract features at a window level. However, unlike jSymbolic, in our implementation, the window length is specified in musically relevant units such as score measures rather than seconds. This provides more pertinent information for processing music scores.

In contrast to other tools, our solution provides an out-of-the-box capability for processing large corpora through parallel processing, resulting in a reduction of the required time.

The design principles and the features included in musif were presented in a previous publication [20]. The code and documentation of musif is available online [3] .

3. BENCHMARKING METHODOLOGY

To assess the performance of musif in comparison to other tools, we devised a benchmarking methodology. Initially, we identified several datasets that enable testing of diverse file formats. Subsequently, we developed a standardized protocol based on an AutoML pipeline [21]. We evaluated the computational resources utilized by each tool during extraction and their respective efficacy in various classification tasks.

3.1 Datasets

We selected five datasets to evaluate the performance of the tools in analyzing both Standard MIDI Files (SMFs) and highly informative music score formats. For MIDI analysis, we aimed to test both music scores and performances. As for highly informative file formats for music scores, we chose MusicXML and **kern due to their popularity, availability of large datasets, various conversion tools, and compatibility with common music score editing software such as Finale, Sibelius, and MuseScore. While MEI was considered as an option, the limited availability of datasets in this format led us to leave it for future studies.

In this study, we considered the following datasets:

- **ASAP** [22]: This dataset contains music performances derived from the Maestro dataset [23] and is synchronized with a corresponding score obtained from the MuseScore's crowd-sourced online library. The dataset comprises 222 music scores in MusicXML and MIDI formats, as well as 1068 music performances in MIDI format. The authors have rectified any significant notation errors found in the music scores. We used this dataset for composer recognition based on music scores and music performances.
- **EWLD** [24]: It contains lead sheets obtained from Wikifonia, a crowd-sourced archive. To reduce errors in music score transcription by inexperienced users, the authors applied algorithmic selection criteria to the dataset.

[3] `https://github.com/DIDONEproject/musif`, `https://musif.didone.eu`

Table 2. Resulting task size for each dataset and feature set.

Extension	Dataset	Classification task	Samples	Classes	Features				
					musif		music21		jSymbolic
					musif	musif native	music21	music21 native	
MIDI	*ASAP performances*	Composer	211	10	710	91	633	602	225
	ASAP scores	Composer	211	7	710	91	633	602	225
	EWLD	Genre	2645	11	710	91	633	602	225
	JLR	Attribution	109	3	732	113	633	602	226
	Quartets	Composer	363	3	1593	974	633	602	225
	Didone	Decade	1622	8	745	126	633	602	225
MusicXML	*ASAP scores*	Composer	211	7	710	91	633	602	
	EWLD	Genre	3197	11	724	105	633	602	
	JLR	Attribution	109	3	739	120	633	602	
	Didone	Decade	1636	8	971	352	633	602	
***kern*	*Quartets*	Composer	363	3	734	115	633	602	

Specifically, they retained only scores with simple notation, without modulations and with a single melodic part. Moreover, all scores contained key signatures and chords throughout. The dataset was augmented by incorporating genre and composer details, as well as the year of first performance, composer birth and death dates, precise title, and additional metadata. This was achieved by cross-referencing the dataset with information sourced from `secondhandsong.com` and `discogs.com`. We used this dataset for genre recognition.

- **Josquin-La Rue** [25]: This dataset was created within the context of the Josquin Research Project and includes 59 Josquin duos and 49 duos by La Rue. The musical scores underwent a meticulous musicological transcription process. Moreover, the music scores were assigned to two labels based on the security of the attribution, thus resulting in four labels (Josquin secure, La Rue secure, Josquin not secure, La Rue not secure). The musical scores are provided in various file formats including MIDI, MusicXML, **kern, Sibelius, and PDF. We used this dataset for composer classification in a real-world attribution problem.
- **Quartets** [26]: We retrieved a selection of files from the `kern.humdrum.org` website, consisting of all available string quartets in **kern format by Mozart, Haydn, and Beethoven. While the original sources of these musical scores are not always declared, the encoding quality is generally considered to be at a musicological level. In total, we obtained 363 files. We used this dataset for composer classification.
- **Didone** [16]: With the aim of filling an under-studied repertoire, we curated, analyzed, and transcribed over 1600 arias from 18th-century opera, written by dozens of composers. The music scores were transcribed into MusicXML format using Finale Music software and revised by three musicologists independently. Harmonic analyses were added by expert musicologists using MuseScore software in accordance with a prior standard [12, 13] and were reviewed automatically using the ms3 tool [27]. We also included various metadata in the database such as year and place of premiere, composer, and high-level formal analysis. This database is an ongoing project and will be made freely available in 2024. We utilized this dataset for classifying the period of composition of each piece, each period being defined in decades (i.e., 1720s, 1730s, 1740s, etc.).

3.2 Experimental setup

After selecting the datasets, a standardized protocol was developed for benchmarking the three aforementioned tools. The protocol is based on an AutoML pipeline [21] and comprises the following steps:

1. **Conversion to MIDI**: The datasets were selected and subsequently converted into MIDI format, resulting in two or three file formats for each dataset: MIDI and either MusicXML or **kern. This step aims to evaluate the impact of notational file formats, such as MusicXML or **kern, on classification tasks. Indeed, although MIDI has limited capacity for representing notational aspects of music, it remains uncertain the extent to which these aspects can determine the accuracy of machine-learning algorithms for music symbolic analysis. MusicXML files were converted using MuseScore 3, and **kern files were processed with the Humdrum toolkit [4].
2. **Feature extraction**: Features were extracted from MIDI, MusicXML, and **kern files using the methods detailed in Section 2 with default settings and without the use of windows, resulting in one array of features for each file. The purpose of this step was to measure the computational cost of the tools. Therefore, all available files in the datasets were used to obtain a larger number of samples and a more accurate estimation of the computational cost, even if they were discarded in later steps. For instance, MIDI scores were already provided in the ASAP dataset; however, we additionally converted them from the MusicXML files. As a result, we extracted features from more files than necessary. We created a CLI tool in Python for music21 while we utilized the official CLI tools for jSymbolic and musif. Each file format was processed individually, resulting in CSV files for each format. We calculated the average time and RAM usage of each tool. Furthermore, CPU time was collected as a measure of the required time without parallel processing. Lastly, we documented the number of files for which each tool produced errors.
3. **AutoML**: A state-of-the-art machine learning approach was employed using the Python module `auto-sklearn` [21]. The method utilizes Bayesian

[4] See footnote 2.

Table 3. Accuracies of AutoML using 10-fold cross-validation on the first ten principal components. The best-performing tool is underlined. The best-performing combination is shown in bold.

Extension	Dataset	Dummy guessing	Tools					Combinations			
			musif	musif native	music21	music21 native	jSymbolic	musif native + music21 native	musif native + jSymbolic	music21 native + jSymbolic	musif native + music21 native + jSymbolic
MIDI	ASAP performances	.100	.960	.715	.978	.976	.916	.972	.962	**.980**	.979
	ASAP scores	.146	.743	.644	.781	.751	.780	.791	.819	.819	**.857**
	EWLD	.091	.201	.157	.212	.204	.257	.219	.245	.242	**.259**
	JLR	.344	.700	.642	.779	.751	.722	.711	**.751**	.742	.741
	Quartets	.340	.678	.668	.725	.711	.810	.768	**.831**	.791	.822
	Didone	.125	.359	.362	.403	.380	.443	.414	.451	**.479**	.462
MusicXML	ASAP scores	.171	.773	.669	.759	.745		**.785**			
	EWLD	.091	.216	.185	.215	.201		**.231**			
	JLR	.334	**.793**	.663	.768	.756		**.793**			
	Didone	.126	.398	**.399**	.384	.374		.392			
**kern	Quartets	.340	.713	.711	.767	.763		**.810**			

optimization with surrogate models based on random forests and generates ensembles of models by exploring a vast array of possible architectures. 10-fold cross-validation was used, and the balanced accuracy averaged across the test folds was observed. The best-performing model's result was used for comparison. To initiate the AutoML process, a list of valid files for each dataset was initially defined, discarding those processed in the previous step but unsuitable for validating the classification task. Subsequently, files were selected for which all tools succeeded in extraction, creating comparable datasets for validation. Finally, classes with a number of samples less than twice the number of cross-validation splits were eliminated from each dataset. Consequently, the number of files and categories used in our study differs from the numbers officially provided by each dataset. The classification task performed depended on the dataset, as shown in Table 2.

We conducted two primary experiments: one utilizing all of the extracted features and another using only the first ten principal components. To achieve this, we standardized the features and applied PCA to obtain the ten first principal components. The rationale for the latter experiment is that a larger feature space typically requires a longer AutoML optimization process and affects the performance of the trained classifiers. As the tools extract varying numbers of features, this experiment enables a principled comparison of the usefulness of the non-redundant information generated by the different tools by homogenizing the number of variables in the AutoML process. In other words, it helps decouple the AutoML optimization capabilities from the number of features.

Due to the overlap between the features extracted with musif and those with music21 with jSymbolic, we also analyzed the concatenation of music21, jSymbolic, and our features. We also observed the performance of musif and music21 when only the native features were used, i.e. when musif was utilized without music21 features and when music21 was run without jSymbolic features. In the following, we denote these feature sets as "native". We run each feature extraction and AutoML experiment on a Linux machine with 32 GB of RAM and an i7-8700 CPU, ending the AutoML procedure after 30 minutes. We also experimented with longer AutoML processes and more powerful machines for the first 5 columns of tables 4 and 3, but we noticed no significant change in accuracy.

4. RESULTS

Table 1 summarizes the comparative computational efficiency of the three tools. It is observed that jSymbolic outperforms the other tools when no parallel processing is employed. This can be attributed to the superior performance of Java language, which facilitates faster I/O operations and parsing of byte-level structures such as MIDI files. musif's caching system significantly reduces the time required for feature extraction during multiple runs, such as those performed during the development and debugging of newly added features. For MIDI files, the extraction process can be accelerated by a factor of five. When comparing the time needed for extraction, jSymbolic is still faster than musif. However, our caching system is advantageous when a cache is available. Regarding MusicXML and **kern files, musif and music21 use the same parser engine, making their time values more comparable. In this case, music21 is slightly faster than musif but also attempts to extract a smaller number of features. Nevertheless, musif's the caching system allows for a 50% reduction in extraction times. The music21 tool proves to be the optimal choice when taking into account RAM utilization.

Table 2 presents the dataset sizes used in our experiments, which are obtained through the protocol detailed in Section 3.2. The sample sizes vary from 109 to 3197, while the number of classes ranges from 3 to 11, depending on the dataset. The music21 feature extraction process produces a fixed set of 602 native features, supplemented by an additional 31 features re-implemented from the jSymbolic feature set. In contrast, jSymbolic consistently extracts a set of 225 features with minor variations. musif extracts a variable number of features depending on its ability to parse different music structures, ranging from 91 to 974 extracted features. The remaining features extracted by musif are computed using the music21 feature extraction methods. It is worth noting that music21 always converts non-computable features to zero, whereas musif allows users to assign different values or perform other operations.

Tables 3 and 4 demonstrate the effectiveness of feature sets in representing significant aspects of music analysis across various repertoires. The results in Table 4 must be interpreted with caution due to the longer AutoML process required by accurate models when using a higher number of features. Overall, music21 and jSymbolic are effective tools for extracting features from MIDI files, while musif

Table 4. Accuracies of AutoML using 10-fold cross-validation on all the extracted features. The best-performing tool is underlined. The best-performing combination is shown in bold.

Extension	Dataset	Dummy guessing	Tools					Combinations			
			musif	musif native	music21	music21 native	jSymbolic	musif native + music21 native	musif native + jSymbolic	music21 native + jSymbolic	musif native + music21 native + jSymbolic
MIDI	ASAP performances	.100	.983	.839	.983	.984	.985	.983	.985	**.990**	.988
	ASAP scores	.146	.843	.626	.877	.887	.886	.911	.898	**.912**	.937
	EWLD	.0912	.224	.180	.249	.227	.248	.236	.250	.249	**.251**
	JLR	.344	.746	.697	**.806**	.761	.747	.789	.787	.751	.774
	Quartets	.340	.828	.771	.843	.813	.901	.843	.896	.880	**.904**
	Didone	.125	.480	.429	.525	.508	.586	.515	.572	**.596**	.557
MusicXML	ASAP scores	.171	.830	.710	**.880**	.841		.847			
	EWLD	.091	.251	.200	**.266**	.253		.245			
	JLR	.334	.797	.704	**.815**	.806		.750			
	Didone	.126	.510	.504	.527	.516		**.535**			
**kern	Quartets	.340	.822	.786	.830	.820		**.842**			

Table 5. Accuracies of AutoML. Effect of harmonic features on the Didone dataset.

	Extension	Harmonic features	musif	musif native	musif native + music21 native	musif native + jSymbolic	musif native + music21 native + jSymbolic
First 10 PCs	MIDI	No	.359	.362	.414	.451	.462
		Yes	**.380**	.372	.398	.452	**.465**
	MusicXML	No	.398	.399	.392		
		Yes	.385	**.406**	**.409**		
All features	MIDI	No	**.510**	.504	.515	**.596**	.557
		Yes	.507	.437	.518	.575	.560
	MusicXML	No	.480	.429	.535		
		Yes	**.535**	.521	**.564**		

shows promising results for MusicXML files, particularly when utilizing the first ten principal components during validation. This difference in performance can be attributed to the presence of highly correlated features in musif, a consequence of its granularity. We also evaluated combinations of feature sets and found that optimal performance is achieved by employing multiple tools. For MIDI files, jSymbolic is fundamental in achieving model accuracy, but incorporating musif and music21 generally enhances performance. For MusicXML and **kern files, leveraging both musif and music21 yields optimal results, especially when considering the first ten principal components.

When comparing the efficacy of models trained on MusicXML, **kern, and MIDI files, no discernible pattern emerges indicating the superiority of highly informative file formats over SMFs for representing music scores. In fact, the only instances where the MusicXML files exhibit superior performance are in the Josquin-La Rue dataset and genre recognition on the EWLD dataset when all features are utilized. However, for all the remaining tasks, MIDI files demonstrate superior performance. This is likely due to the fact that jSymbolic can only extract features from MIDI files and is simultaneously the most important source of features for music score analysis. Consequently, in this study, the MusicXML and **kern datasets lack some relevant features that can be extracted only when converted to MIDI. Even when comparing only the proposed tool and music21's performances, MusicXML and **kern files do not show a clear advantage over MIDI files, particularly when considering the combination of both tools. It should be noted that jSymbolic can extract features from MEI as well, thus potentially allowing for better performances.

The effect of missing values on tool performance is a significant concern and may be a contributing factor to the comparatively lower results for MusicXML and **kern files. While music21 substitutes all missing values with 0, musif utilizes a hybrid strategy that entails either removing a row or column from the table (refer to Section 2). The most effective method for handling missing values remains an open issue.

We assessed the impact of harmonic features on the Didone dataset using musif. Unfortunately, due to the time-consuming nature of harmonic annotations, we were unable to evaluate these features on the other datasets used in this study. We annotated our dataset of more than 1600 opera arias using the standard established in previous works (see Section 2) and extracted melody- and accompaniment-related features with respect to the local key. The extraction of harmonic features resulted in 22 additional features beyond the 126 listed in Table 2 for MIDI files. For MusicXML files, we extracted 265 additional features, raising the total number of extracted features to 617. We observed an overall improvement in classification accuracy when incorporating harmonic features, as demonstrated in Table 5. The only instance where performance was degraded by the inclusion of harmonic features was for MIDI files when all the available features were considered (without PCA). We interpret this degradation as an indication that longer processing times are necessary for AutoML when additional, possibly highly correlated features are introduced.

5. CONCLUSION

This paper presents a comprehensive analysis of tools for extracting features from symbolic music. A strict protocol was defined to compare the tools in terms of efficiency and efficacy across various repertoires and file formats. The results indicate that using multiple tools is the most effective approach, with the optimal tool choice depending on the file format and repertoire.

The study emphasizes the importance of using file formats that are accessible by multiple tools. However, it remains open whether highly informative file formats such as MusicXML, **kern, or MEI are relevant for the automatic classification of symbolic scores. The available set of features indicates that, while these formats remain fundamen-

tal for certain types of musicological research, they do not seem to entail a significant advantage for machine learning tasks.

The problem of NaN values in extracted features from music scores remains unresolved. Further research is required to explore optimal approaches for replacing, removing, or inferring missing values in music applications.

Additionally, the new musif tool was proposed, which can process various file formats using the music21 parsing engine. The tool also includes a caching mechanism to speed up feature development. Moreover, motivated by the experiments presented in this work, we included the whole music21 and jSymbolic tools in the newer versions of musif, easing the extraction of the combined feature sets from large corpora.

6. ACKNOWLEDGEMENTS

This publication is a result of the Didone Project, which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program, Grant agreement No. 788986. It has also been conducted with funding from Spain's Ministry of Science and Innovation (IJC2020-043969-I/AEI/10.13039/501100011033).

Part of the computational experiments were run at the FinisTerrae III cluster of the Galician Supercomputing Center (CESGA). The authors gratefully acknowledge the access to these resources.

7. REFERENCES

[1] K. C. Kempfert and S. W. K. Wong, "Where Does Haydn End and Mozart Begin? Composer Classification of String Quartets," *Journal of New Music Research*, vol. 49, no. 5, pp. 457–476, Oct. 2020.

[2] W. Herlands, R. Der, Y. Greenberg, and S. Levin, "A Machine Learning Approach to Musically Meaningful Homogeneous Style Classification," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, Jun. 2014.

[3] J. Qiu, C. L. P. Chen, and T. Zhang, "A Novel Multi-Task Learning Method for Symbolic Music Emotion Recognition," 2022.

[4] M. Zeng, X. Tan, R. Wang, Z. Ju, T. Qin, and T.-Y. Liu, "MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training," *FINDINGS*, 2021.

[5] D. Jeong and J. Nam, "Note Intensity Estimation of Piano Recordings by Score-informed NMF," 2017, p. 8.

[6] F. Simonetta, S. Ntalampiras, and F. Avanzini, "Acoustics-Specific Piano Velocity Estimation," in *Proceedings of the IEEE MMSP 2022*, 2022.

[7] H. Vinet, "The Representation Levels of Music Information," in *Computer Music Modeling and Retrieval*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 193–209.

[8] F. Simonetta, S. Ntalampiras, and F. Avanzini, "Multimodal Music Information Processing and Retrieval: Survey and Future Challenges," in *Proceedings of 2019 International Workshop on Multilayer Music Representation and Processing*. Milan, Italy: IEEE Conference Publishing Services, 2019, pp. 10–18.

[9] ——, "Audio-to-Score Alignment Using Deep Automatic Music Transcription," in *Proceeddings of the IEEE MMSP 2021*, 2021.

[10] A. Llorens and A. Torrente, "Constructing *opera seria* in the Iberian Courts: Metastasian Repertoire for Spain and Portugal," *Anuario Musical*, vol. 76, pp. 73–110, Jul. 2021.

[11] F. Moss, W. Fernandes de Souza, and M. Rohrmeier, "Harmony and Form in Brazilian Choro: A Corpus-Driven Approach to Musical Style Analysis," *Journal of New Music Research*, vol. 49, pp. 416–437, 2020.

[12] M. Neuwirth, D. Harasim, F. C. Moss, and M. Rohrmeier, "The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets," *Frontiers in Digital Humanities*, vol. 5, 2018.

[13] J. Hentschel, M. Neuwirth, and M. Rohrmeier, "The Annotated Mozart Sonatas: Score, Harmony, and Cadence," *Transactions of the International Society for Music Information Retrieval*, vol. 4, no. 1, pp. 67–80, May 2021.

[14] C. McKay, J. Cumming, and I. Fujinaga, "jSymbolic 2.2: Extracting Features from Symbolic Music for Use in Musicological and MIR Research," in *Proceedings of the 19th International Society for Music Information Retrieval Conference*, 2018, pp. 348–354.

[15] M. S. Cuthbert, C. Ariza, and L. Friedland, "Feature Extraction and Machine Learning on Symbolic Music Using the Music21 Toolkit," in *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011*. University of Miami, 2011, pp. 387–392.

[16] A. Torrente and A. Llorens, "The Musicology Lab: Teamwork and the Musicological Toolbox," in *Music Encoding Conference Proceedings 2021, 19–22 July, 2021 University of Alicante (Spain): Onsite & Online, 2022, ISBN 978-84-1302-173-7, págs. 9-20*. Universidad de Alicante / Universitat d'Alacant, 2022, pp. 9–20. [Online]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=8463477

[17] C. McKay and I. Fujinaga, "jSymbolic: A Feature Extractor for MIDI Files," in *Proceedings of the 2006 International Computer Music Conference, ICMC 2006, New Orleans, Louisiana, USA, November 6-11, 2006*. Michigan Publishing, 2006. [Online]. Available: https://hdl.handle.net/2027/spo.bbp2372.2006.063

[18] M. S. Cuthbert and C. Ariza, "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data," *International Society for Music Information Retrieval Conference (ISMIR 2010)*, pp. 637–642, 2010. [Online]. Available: http://ismir2010.ismir.net/proceedings/ismir2010-108.pdf

[19] C. L. Krumhansl, *Cognitive Foundations of Musical Pitch*. Oxford University Press, 1990.

[20] A. Llorens, F. Simonetta, M. Serrano, and Á. Torrente, "Musif: A Python Package for Symbolic Music Feature Extraction," in *Proceedings of SMC 2023 - Sound and Music Computing Conference*, Stockholm, 2023, June, pp. 132–138.

[21] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning," arXiv, Tech. Rep. arXiv:2007.04074, Sep. 2021. [Online]. Available: http://arxiv.org/abs/2007.04074

[22] F. Foscarin, A. McLeod, P. Rigaux, F. Jacquemard, and M. Sakai, "ASAP: A Dataset of Aligned Scores and Performances for Piano Transcription," in *Proceedings of the 21st International Society for Music Information Retrieval*, 2020, Proceedings.

[23] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. H. Engel, and D. Eck, "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset." in *International Conference on Learning Representations*, 2019.

[24] F. Simonetta, F. Carnovalini, N. Orio, and A. Rodà, "Symbolic Music Similarity through a Graph-based Representation," in *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion - AM'18*. ACM Press, 2018.

[25] J. Cumming, C. McKay, J. Stuchbery, and I. Fujinaga, "Methodologies for Creating Symbolic Corpora of Western Music Before 1600," in *Proceedings of the ISMIR*. Paris, France: ISMIR, Sep. 2018, pp. 491–498.

[26] C. S. Sapp, "Online Database of Scores in the Humdrum File Format," in *Proceedings of the 6th International Conference on Music Information Retrieval*, 2005, p. 2.

[27] J. Hentschel and M. Rohrmeier, "Creating and Evaluating an Annotated Corpus Using the Library Ms3," 2020.