Arty: Expressive timbre transfer using articulation detection for guitar

by

Sebastian Franjou

B.S. Music and Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
September 5, 2022

Certified by .....
Eran Egozy
Professor of the Practice in Music Technology
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Arty: Expressive timbre transfer using articulation detection for guitar

by

Sebastian Franjou

Submitted to the Department of Electrical Engineering and Computer Science
on September 5, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this work, we propose a novel approach to timbre transfer. Timbre transfer is the transformation of an instrument's timbre to match the timbre of another instrument while preserving key musical information like pitch and loudness. Current attempts tend to rely either on MIDI pitch and velocity information, or on Deep Learning networks. The former approach requires discarding a lot of information and hence suffers from a loss of expressivity, while the latter results in expressive but unstable and difficult to tune systems.

Arty aims to address this problem by adding expression data to the collected MIDI. By detecting instrument-specific playing techniques called articulations, and transcribing these articulations as MIDI data, Arty attempts to provide an expressive yet flexible alternative to the methods above for timbre transfer from guitar. The use of MIDI allows for integration with other music performance systems and doesn't impose a particular sound synthesis method.

We created a new dataset, the Arty dataset, and used it in conjunction with existing data to train a model to classify right-hand and left-hand guitar playing techniques. We implemented a website as a user interface to allow users to easily convert their guitar playing to MIDI. Arty achieved fairly high accuracy on the dataset, but the user study showed that Arty's real world accuracy is much lower, in part because real-world data is different from and more diverse than our dataset. The user study did however reveal a strong interest for such a system from advanced virtual instruments users.

Thesis Supervisor: Eran Egozy
Title: Professor of the Practice in Music Technology

Acknowledgments

To Amelia Paine, for being by my side the whole time I was writing this thesis.

To Anna Devillaire, for finding the worst possible time to visit: it made the last week of writing before the deadline a little more stressful, but a lot more fun.

To the MIT Video Game Orchestra exec, for the late night hangs and group lunches in storage; special thanks to Joey Gu for being a great friend for the four years we've run this orchestra together.

To all the MIT music department, and in particular: Marcus Thompson, for putting up with me despite me missing auditions, turning in programs late, injuring myself, etc... Emily Pollock, for being the best academic advisor and teaching the best seminar; Fred Harris, for your unparalleled and inspiring dedication to helping all of your students thrive and for all the help you've given me over the years.

To my labmates Madi Wang and Matt Caren: we may have failed to acquire a projector screen, couch and minifridge for lab before graduating, but we had fun thinking about doing so together.

To Carlos Alvarado for always being the first tester for all my projects, and for not complaining while I debug my code in front of him and make him stay for three times longer than he signed up for.

To the Virtual Orchestration Facebook group for being so enthusiastic about Arty and trying it out.

To Ryaan Ahmed: it was great to help teach Interactive Music Systems with a fellow nylon strings plucker, and I both enjoyed and learned a lot from our conversations.

And of course, thanks to Eran Egozy for being an amazing mentor, great to work with and to hang out with, and truly passionate about music tech. Thank you for taking me as a TA and then as an MEng student, even while being on sabbatical!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital emulation of musical instruments is ubiquitous in modern music, but even as synthesis technology improves, expressivity remains a challenge. Although playing an instrument always involves controlling the pitch, loudness and duration of notes, performers also control many other parameters that contribute significantly to the music. These parameters are often called articulations, and although some are shared across a wide variety of instruments, others are specific to certain instruments because they refer to specific playing techniques. For example, a violinist might adjust the direction of their bow, whereas a guitarist might might dampen strings with their palm. Because of how numerous and specific they are, these parameters are usually difficult to incorporate into audio synthesis.

Although articulations in the narrow sense only refer to the way notes start, end, and are separated from each other [2], the term in its general sense also applies to many other musical parameters, especially parameters that vary over the course of a single note. In classical music, articulations often have a name and specific symbol in music notation. Many deal with the duration and separation of notes, like staccato or legato. Others deal with dynamics within the span of a single note, like sforzando or swells. Articulations are a crucial part of a good musical performance, and because each instrument uses specific techniques to execute them, they contribute to the instrument's specific sound. However, despite the musical importance of articulations, synthesis systems often offer limited control over them, because of the

complexities and cost associated with making a synthesizer with these parameters, and the complexity of programming[1] them for the user.

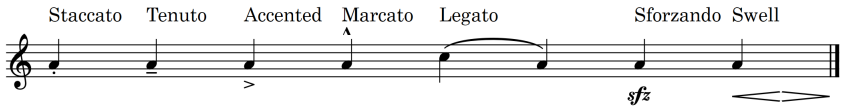


Figure 1-1: Examples of common articulations

Most synthesizers are designed to be controlled from a piano-like MIDI keyboard, using MIDI Note On and Note Off messages. Some models include two additional continuous parameters: pitch bend and modulation values. However, this control protocol does not convey enough information to properly specify articulations or playing techniques. Higher end synthesis systems use additional MIDI Control Change (CC) messages[2] or keyswitches[3] to convey this information, but this additional layer of complexity often prevents the direct translation of a keyboard performance into a convincing equivalent on violin, clarinet, etc... This expressivity must be added after the fact.

One way to remedy this problem is to forgo using a keyboard in favor of a more expressive controller. Devices like the Electronic Wind Instrument (EWI)[4] or the Roli Seaboard[5] use additional sensors to obtain more information from the performer, which can then be mapped to additional synthesis parameters such as articulation. However, even though these controllers sometimes emulate existing instruments, the way they are played is quite different from the instruments they take inspirations from. For example, the EWI uses a bite sensor, which doesn't correspond to how wind instruments are played, and the Roli Seaboard senses the height at which the

[1]In this context, MIDI programming means inputting MIDI information such as notes, control change messages and articulation switches to achieve an expressive performance. It usually involves editing MIDI in a Digital Audio Workstation and/or specialized hardware such as MIDI controllers with a modulation wheel.

[2]MIDI Control Change messages are a special kind of MIDI messages that allow for the continuous control of a parameter over time.

[3]Keyswitches are the use of specific MIDI pitches as toggle switches for parameter.

[4]https://www.akaipro.com/products/ewi-series

[5]https://roli.com/products/seaboard

key is pressed, unlike a regular piano keyboard. These differences are often necessary to offer additional dimensions of control and expressivity, but they make controlling these parameters a skill that must be learned and practiced, similarly to learning an instrument.

Ideally, a performer would be able to play their primary instrument and use it as a controller for the synthesis of other instruments. Expressivity requires proficiency, and using any other controller would require becoming sufficiently proficient at it to be able to convey all the desired articulations during the performance. Since musicians are already proficient on their instrument, allowing them to use it as a the controller for generating articulation data would be ideal, provided we can gather such data from a performance.

One way to state this idea is as the task of changing the timbre of one instrument into another. This task is known in the literature as timbre transfer ([3]).The current state-of-the-art in timbre transfer uses deep learning to both analyze the original audio of the instrument being played and generate the transformed timbre. Although this approach is very powerful because it can potentially take into account all aspects of the original audio, deep learning models are computationally expensive to train and run, and they lack the widespread compatibility and ease of manipulation that MIDI data offers. Most notably, the sound quality doesn't match what current professional synthesis software can achieve, and the system overall integrates poorly with a modern music production workflow.

Arty is timbre transfer software that aims to offer the expressivity of deep-learning based timbre transfer while retaining the compatibility and ease of MIDI editing. While the larger goal is to allow timbre transfer from any instrument to any other, in this thesis, we focus only on guitar as the input instrument. To achieve this goal, Arty can detect pitch, loudness, and playing technique from guitar audio, and transcribe them as MIDI Note On, Note Off, and Control Change messages. This MIDI can then be used to drive a high-end third-party synthesis engine. The MIDI CC messages triggers articulation changes in the synthesis engine, keeping the expressivity of the original performance. Rather than synthesize new audio from scratch, Arty generates

a fully expressive MIDI input sequence for a third party synthesizer.

We detect playing techniques rather than articulations directly to allow for more explicit control over the output, since classifying guitar articulations can be ambiguous even for a human listener. For example, the difference between a regular short note and a staccato is not always clearly defined, but it is common for software synthesizers to have a separate staccato option. By detecting a playing technique, for example palm mute, which is unambiguous to a human listener, we can increase the likelihood that the generated midi corresponds to the player's intention.

This approach has several benefits: it allows users to input complex articulations at a much faster pace, while allowing them to keep using their synthesis engine of choice. It also allows them to edit the MIDI data just like they would any other MIDI data, using the tools they already know. Arty's interface is simple, and assuming the user is proficient at guitar, Arty can speed up the MIDI input workflow without the user having to invest time learning a complex new tool. The musical complexity all comes from the original guitar performance, and not from having to manipulate Arty's parameters, or manually input MIDI control changes for articulation switches. In short, Arty allows users do timbre transfer simply by playing their instrument, without sacrificing the ability to edit and fine-tune the result using the tools they are familiar with.

Chapter 2

Literature Review

2.1 Guitar controlled synthesizers

Attempts to make a guitar sound like other instruments by using it as a controller for synthesis date back to at least 1977 with Roland's GR500, using CV/Gate as a control scheme [4]. MIDI guitars, which allowed for use with third party synthesizers, came soon after, with the Roland GR700 in 1985, and then the GK1 special MIDI pickup in 1986 [5]. However all of these require special hardware instead of relying on the acoustic sound produced by an instrument, and are limited to pitch and loudness processing. Modern guitar synthesizers such as Electro Harmonix Organ 9 [5] are capable of transcribing pitch bends and sometimes vibrato without relying on additional sensors, but don't use a specific representation for specific playing techniques (or, in the case of closed source code, might do so internally but don't communicate it via MIDI or another universal communication protocol).

2.2 Playing technique classification

The commercial products mentioned above focused on transcribing pitch in real-time, often in integrated hardware devices, with little to no focus on detecting playing techniques. However, there has been some academic research on playing technique detection, often with the goal of automating transcription of recorded music into

sheet music. Some attempts use additional sensors to gather more information to help detect the playing technique, as opposed to relying on audio only. Peiper et al use sensors on a violin's bow to complement audio data, and then train a decision tree to select a playing technique for the currently played note [6].

2.2.1 Trained classifiers

Most attempts rely on the audio alone, often training a classifier on labelled data. Generating enough audio data to train a complex model can be costly. In [7], the authors' solution is to mix real recorded audio with synthesized clips generated from sample libraries. They use this approach to train a fully connected CNN to detect playing techniques for various Chinese bowed instruments from a Mel-Spectrogram.

Another way to circumvent having to make and annotate a large dataset is to use a less complex model, which requires less data to train. The trade-off with simpler classification models is that more audio pre-processing must be done to provide it with more meaningful input features. This is the approach taken in [8]. A large number of timbral features are used, which are then simplified by the use of a sparse coding technique (here LASSO), and fed into one Support Vector Machine per playing technique.

2.2.2 Hand-tuned algorithms

A different approach is to forgo training a classifier entirely, and hand tune parameters in a specifically designed detection algorithm. Although used by Ozaslan et al in [9], this approach requires designing a new algorithm for each playing technique. As such, they limited their approach to articulations happening at the attack of the note, and only worked on nylon string guitar (as opposed to including steel string and electric guitar).

2.2.3 Hybrid algorithms

A hybrid approach is taken in [10]. Similarly to [8], timbral features are used as inputs to SVMs to determine playing techniques, but hand tuned algorithms eliminate implausible candidates using pitch information. This approach is robust enough to yield good results on guitar solos taken from commercial recordings, detecting hammer-on, pull-off, slide, bend, and vibrato.

Similar approaches are taken in [11] and [1] for bass guitar and guitar respectively, using *Inertia Ratio Maximization using Feature Space Projection* as a feature reduction algorithm. The authors made their dataset available[1], although few algorithms use the same dataset, making comparisons difficult. The make a distinction between right-hand (fingerstyle, picked, palm-mute) and left-hand techniques (bend, vibrato, slide, harmonics, dead note), and detects both independently.

A more recent attempt is [12], where extensive use of pitch information is made to detect pitch-based articulations, and Convolutional Neural Networks are used on an MFCC to detect articulations. The particular CNNs model used can only take in fixed length inputs however, so the authors focus on classifying transitions between notes (hammer-on , pull-off, slides) and pitch-based playing techniques (bends). This approach is rather successful, but limited in the kind of techniques it detects.

2.3 End-to-end timbre transfer

More recent work on timbre transfer uses end-to-end deep learning models, as some believe traditional DSP techniques will not achieve high accuracy for complex classification tasks [13]. Chen et al. [14] employ a Generative Adversarial Network using the constant-Q transform of the input audio, taking an approach inspired by works on neural image processing. Other approaches use Variational Auto Encoders with a music-specific decoding architecture to achieve a more interpretable middle representation, although this representation still offers far less control than regular synthesizers do. Bitton et al [15] use a vector-quantized latent space and filtered

[1] `https://www.idmt.fraunhofer.de/en/business_units/m2d/smt/guitar.html`

noise as a synthesis engine in the decoding module. [16] uses a similar approach, but uses the author's DDSP library to use both a filtered noise source and an additive synthesizer, passed through a reverb unit, as their synthesis engine.

Chapter 3

Design

This chapter introduces Arty's intended use case, design goals, and a basic system overview.

3.1 Preamble: Inputting expressive MIDI without Arty

To understand Arty's design goals, we must first understand how inputting expressive MIDI is usually done. The goal is to use virtual instruments (VSTis) to produce an expressive, usually realistic sounding emulation of orchestral instruments. A virtual instrument is a software synthesizer plugin that reads MIDI information and generates the appropriate sound. A single virtual instrument usually emulates a single instrument, so in order to create a complete piece of music, the output of several virtual instruments must be combined together. All of this is done in a Digital Audio Workstation (DAW): software that can route MIDI to virtual instrument plugins, and mix together the resulting sounds sources (see Figure 3-1). Each sound source in a DAW is called a *track*. A typical DAW session for a virtual orchestra would have one virtual instrument per track, and each track would have its own MIDI information, much like each player in an orchestra has their own assigned sheet music.

To create a piece of virtual orchestral music, it is hence necessary to generate the

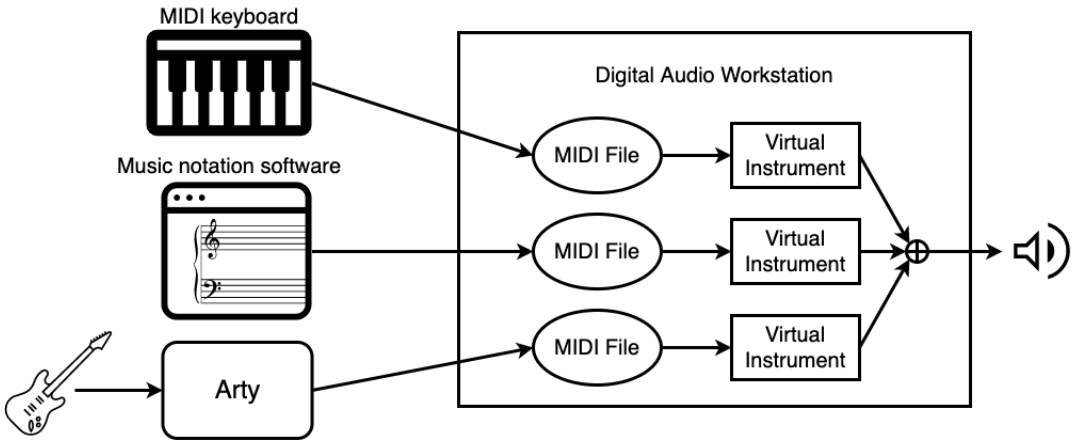


Figure 3-1: Data flow between frontend and backend

MIDI information for each track/instrument. There are 2 main approaches: using notation software, and recording directly into the DAW (see Figure 3-1).

3.1.1 Using Notation software

The first step in entering MIDI information is to record the basic pitch and rhythm information. This can be done by writing music on a digital staff in software like Finale[1], Musescore[2] or Dorico[3] (3-2). Such software usually also allows for entering some additional information: dynamics, articulations, etc... This can all be done using a mouse and computer keyboard, although often MIDI piano keyboards and sometimes specialized hardware such as a stream deck[4] can be used to speed up the process.

Once the score is completed, it can be exported as a MIDI file and used as the input for a virtual instrument. In some notation software such as Dorico[5], the notated articulations and dynamics can be converted into MIDI information that will be reflected in the virtual instrument's playback, using MIDI CC messages, patch change messages, etc...

[1] https://www.finalemusic.com/
[2] https://musescore.org/en
[3] https://www.steinberg.net/dorico/
[4] https://www.nycmusicservices.com/notation-express/
[5] https://steinberg.help/dorico_se/v4/en/dorico/topics/library/library_expression_maps_c.html

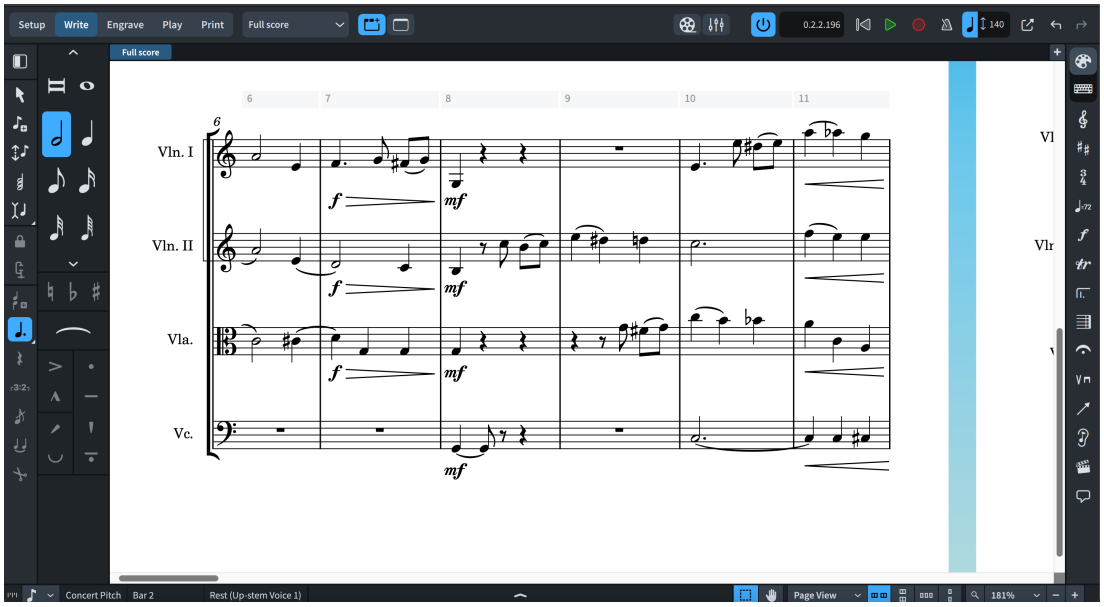
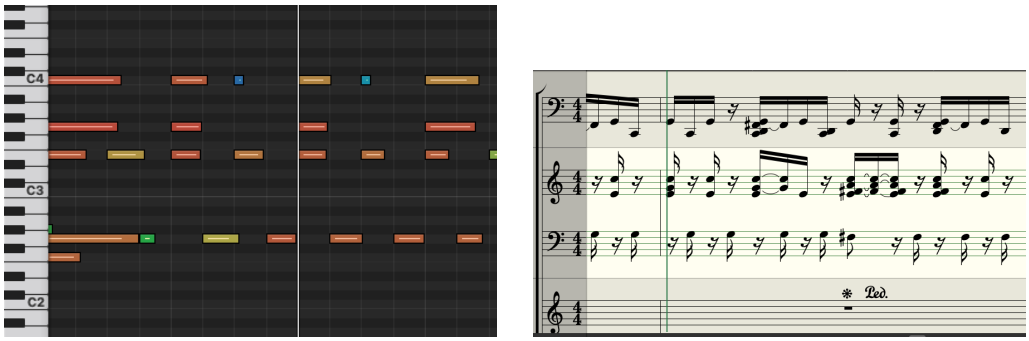


Figure 3-2: Notation software

3.1.2 Using a DAW

It is also possible to enter MIDI information directly into a DAW. Using a MIDI controller (usually a keyboard), one can play notes in real time and have the DAW record their pitch, rhythm, velocity, and any other information provided by the controller. This method effectively transform a piano performance into another instrument, and as such is very efficient for pianists. However, there are articulations that cannot be played on a piano, such as swells or changing from *pizzicato* to *arco* on a violin. To solve this problem, MIDI controllers often have additional continuous controls such as a pitch bend and modulation wheel that can convey continuously changing values, allowing for swells and other smooth variations in sound. Switches between playing techniques (such as *arco* to *pizzicato*) are often achieved with *keyswitches*: playing a key on the keyboard that won't not produce any sound directly, but instead triggers the technique switch in the virtual instrument. The keys chosen for keyswitches are notes that are out of range for the emulated instrument, as not to interfer with the musical performance. However, it can be difficult to control the melody, continuous information with the wheels, and keyswitches all at the same time, so it is not uncommon to first record the melody, and then overdub the extra expressive information in



(a) Piano roll (b) Musical notation editor

Figure 3-3: MIDI editing options in the Logic Pro X DAW

a separate recording round.

DAWs also usually offer a MIDI editor that can be used to edit the MIDI that was recorded, for example to correct mistakes or to add in additional expressive information such as MIDI CC messages. It often takes the form of a piano roll, although it sometimes features a staff and functions as music notation software (see Figure 3-3). This MIDI editor can be used to write in the melody directly, which allows for non-real-time MIDI input without using a separate notation software. However, these editors do not usually have features that map notated articulation to MIDI, so the keyswitches and MIDI CC messages must be added manually.

3.2 Design goals

Arty aims to offer an alternative way to input expressive MIDI, by turning guitar audio into MIDI. In addition to transcribing pitch, rhythm, and loudness, it detects playing techniques and uses them to change articulations in the virtual instrument by triggering keyswitches. Arty is designed to be:

Intuitive

Using Arty should be as intuitive and immediate to guitarists as using a MIDI keyboard is to pianist. There should be no special techniques to learn and no additional practice required to use Arty. The process should be simple for someone familiar with DAWs and virtual instrument. The experience should feel as much as

possible like playing the guitar normally, yet having it sound like another instrument in the recording.

Integrated with pre-existing workflows

Using Arty should be compatible with the workflows described above. In particular, using Arty should not make editing MIDI or mixing more complicated than it is with the above solutions. Expressive MIDI from Arty should be structurally indistinguishable from expressive MIDI inputted using another method, to ensure full compatibility with existing tools.

Accessible

Arty shouldn't require specialized software, and should as much as possible only use hardware the typical Arty user would already own: a guitar and an audio interface. As much as possible, it shouldn't require a specific DAW or virtual instrument type (by being limited only to Kontakt[6] libraries for example).

3.3 System Overview

The user interacts with Arty via a web page. They can record audio directly into the website, rename it, and listen to it before submitting it. Alternatively, they can upload files from their computer. The audio needs to be from an electric guitar plugged directly into an audio interface: a direct, dry guitar signal.

The audio is sent to the Arty server, which first transcribes the notes played, extracting pitch, rhythm, and volume. It then analyses each note, and finds the playing technique used by the right hand (called *excitation style*) and by the left hand (*expression style*).

All this information is used to make a MIDI file that contains MIDI messages corresponding to the notes played and MIDI messages that trigger keyswitches corresponding to the playing techniques detected. Although Arty aims to be compatible with any VSTi with minimal configuration, keyswitches are always targeted to a spe-

[6]Native Instruments Kontakt is a software sampler which is required by many virtual instruments, and offers scripting tools for these virtual instruments. `https://www.native-instruments.com/en/products/komplete/samplers/kontakt-6/`

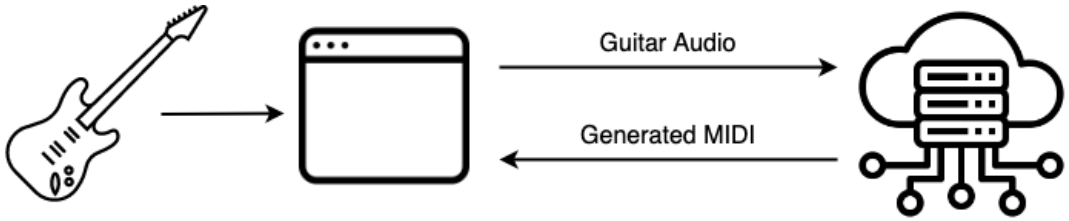


Figure 3-4: Data flow between frontend and backend

cific virtual instrument. For the purpose of our user study, Arty currently maps to the "Strings - Violin Solo 2 KS.sfz" from the Sonatina Symphonic Orchestra Virtual Instrument[7].

This MIDI file is sent back to the user and downloaded onto their machine. The user then opens their DAW, makes a track with the corresponding virtual instrument, and drags and drops the MIDI onto that track. From there, the rest of the process is exactly as it would be if the MIDI was entered any other way.

[7] `https://github.com/peastman/sso`

Chapter 4

User Interface

We implemented a web interface for the algorithm, hosted on Microsoft Azure.

4.1 Front-end

The front-end is written in HTML/javascript, with Bootstrap[1] used for styling (see Figure 4-1). Users can record audio straight into the website. A waveform display provides visual feedback on the audio input. Once a clip is recorded, the user can name it, listen to it, and choose whether to delete or convert it (see Figure 4-2).

Alternatively, users can upload `.wav` files directly to the website using the upload box.

After receiving audio, Arty processes the audio, converts it to `.wav` if needed (using the audioBufferToWav[2] package) and then downloads the corresponding midi file. If no notes are detected in the audio, the website displays an error message (see 4-3).

The user can then drag and drop the MIDI file into their DAW to use with a VSTi.

[1] `https://getbootstrap.com/`
[2] `https://www.npmjs.com/package/audiobuffer-to-wav`

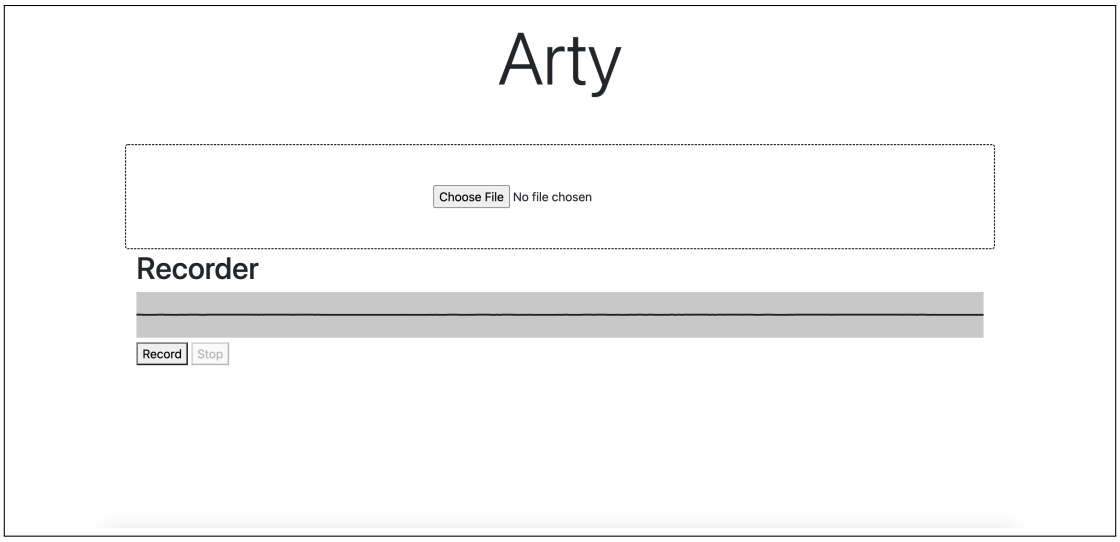


Figure 4-1: Arty website

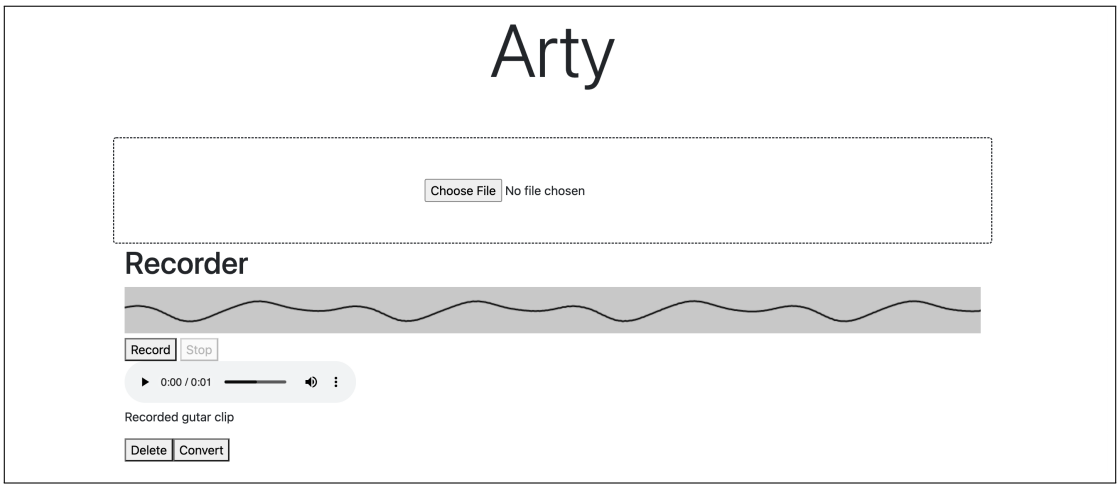


Figure 4-2: Arty website with clip recorded

4.2 Back-end

The back-end is a Flask Python server, hosted on Microsoft Azure. The API has a single endpoint, `/uploadFile/<filename>`. Upon receiving a POST request with a `.wav` file, it downloads the file on the server, processes it, and generates a MIDI file at `/static/generated_files/`. The front-end then downloads the file from that location.

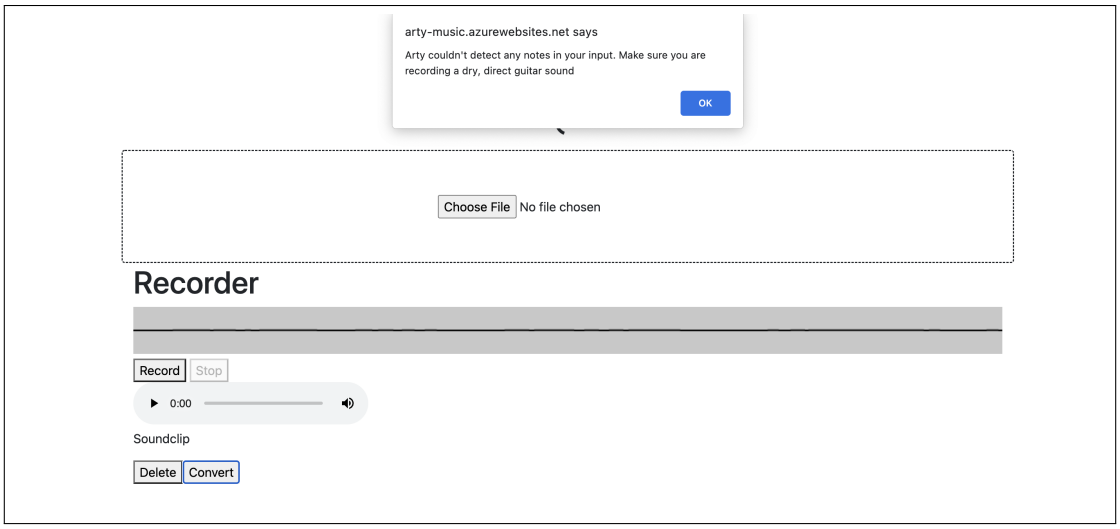


Figure 4-3: Arty website error message on incorrect audio input

4.3 Sound synthesis with third party tools

Arty doesn't directly provide a DAW and virtual instrument to play the generated MIDI file, so third-party tools have to be used. This is by design, as third-party tools are better sounding and easier to work with for experience musicians than what the authors could build in the scope of this work.

We had to chose specific third-party tools for sound synthesis, since Arty generates a MIDI file with articulation change information, and how to trigger these articulation changes depends on the virtual instrument playing the MIDI file. We chose a free and cross-platform virtual instrument to simplify user testing. The generated MIDI files are designed to work with the `"Strings - 1st Violins KS.sfz"` soundfont from Sonatina Symphonic Orchestra[3], which uses keyswitches to trigger articulation changes. This MIDI file should work with most VSTi samplers; we used Plogue Sforzando[4] since it is free and cross-platform.

[3] `https://github.com/peastman/sso`
[4] `https://www.plogue.com/products/sforzando.html`

Chapter 5

Dataset

We detect guitar playing techniques by training a classifier on annotated examples. In order to train the classifier, and to evaluate Arty's classification and transcription accuracy, we use Kehling et al's [1] IDMT-SMT-GUITAR_V2 second dataset, as well as an additional dataset created by the author for this thesis: the Arty dataset.

5.1 IDMT-SMT-GUITAR_V2 dataset

The IDMT-SMT-GUITAR_V2 dataset is a set of annotated electric guitar recordings. The audio was recorded using a variety of guitars and pickups plugged directly into an audio interface. The annotations are `.xml` files with the same name as the corresponding audio file. An `.xml` file contains several note events, each note event containing onset and offset times, pitch, as well as any expression and excitation style information (see Table 5.1 and Appendix A.4 for details). The specification for the dataset also includes a field for vibrato extent and speed. Some of the audio recordings in the dataset are polyphonic, but for this work, we only focus on monophonic detection. After removing the polyphonic examples from the dataset using the annotation dataset, we found that the dataset was a bit unbalanced, with certain combinations of excitation and excitation styles underepresented (see Table 5.2). We therefore constructed a new dataset, the Arty dataset.

5.2 Arty dataset

The Arty dataset is also composed of direct-to-interface electric guitar recordings. Its format follows the format of the IDMT-SMT_GUITAR_V2 dataset (a more extensive description can be found in appendix A). The Arty dataset contains normal, slide and vibrato expression styles, and all excitation styles. The Arty dataset focuses on providing at least one example for those expression/excitation combinations for each pitch, something which the IDMT-SMT-GUITAR_V2 dataset doesn't guarantee (see Table 5.2). To do so, examples of each excitation/expression style pair are recorded on every fret below the 12th fret, on every string of the guitar[1]. This covers most notes played on the guitar, and it also provides all the usual playing locations for each pitch. Since a particular pitch can be played on various strings on a guitar, with various tones, this should help the classifier be more robust in the face of timbre differences for a given pitch.

5.3 Dataset Augmentation

The Arty dataset was all played with the same electric guitar. To introduce more timbre variation and try to limit the model overfitting to one specific guitar, we augmented the Arty dataset using Positive Grid Bias FX 2's *Guitar Match*[2] feature. *Guitar Match* analyzes the timbre of a guitar and processes it to match the tone of another guitar model, although it introduces a fair amount of noise in the process. We augmented the Arty dataset by processing with the *Custom '57 Goldtop Reissue* and *Vintage SHR Antique* settings, to emulate the sounds of Gibson Lespaul and Fender Stratocaster-style guitars respectively, two of the most popular guitar models.

The Arty dataset complements the IDMT-SMT-GUITAR_V2 dataset and is meant to be used in tandem with it.

[1]There are no vibrato examples on open strings and no slides up to notes below the third fret because such combinations aren't playable.

[2]`https://www.positivegrid.com/bias-fx-2-guitar-match`

Style type	**Playing Technique**	**Abbreviation**
Excitation style	Finger-style	FS
	Palm-muted	PK
	Picked	MU
Expression style	No expression style	NO
	Harmonic	HA
	Dead note	DN
	Slide (pitch)	SL
	Vibrato	VI
	Bend	BE

Table 5.1: Annotated playing techniques in the Arty and IDMT datasets

	PK	FS	MU
NO	395	101	282
BE	12	7	9
DN	219	9	9
HA	92	2	9
SL	21	0	12
VI	40	8	21

(a) IDMT-SMT-GUITAR_V2

	PK	FS	MU
NO	89	84	78
BE	0	0	0
DN	0	0	0
HA	0	0	0
SL	60	60	60
VI	72	72	0

(b) Arty dataset

	PK	FS	MU
NO	484	185	360
BE	12	7	9
DN	219	9	9
HA	92	2	9
SL	81	60	72
VI	112	80	21

(c) Combined Datasets

Table 5.2: Occurrences of expression/excitation style pairs in the datasets

Chapter 6

Implementation

All signal processing and classification happens in the Python backend, using the Essentia library [17]. The first step in the audio-to-expressive-MIDI conversion is transcription: segmenting the audio into note events by estimating an onset, offset and pitch for each event. Arty then extract features from each note, and uses them as the input for a classifier. The classifier determines the excitation and expression styles for each note. Finally, a MIDI file is generated which maps these note events, expression and excitation styles to MIDI messages (see Figure 6-1).

6.1 Transcription

The first step in the articulation detection algorithm is to identify individual notes and find their pitch.

6.1.1 F0 estimation

We first apply an equal loudness filter on the audio, then use essentia's implementation of the Yin algorithm[1] ([18]), which computes an estimate of the fundamental frequency as well as a confidence metric.

We then discard all f0 estimates that fall below a confidence threshold by setting them to 0 (see Figure 6-3).

[1] `https://essentia.upf.edu/reference/std_PitchYin.html`

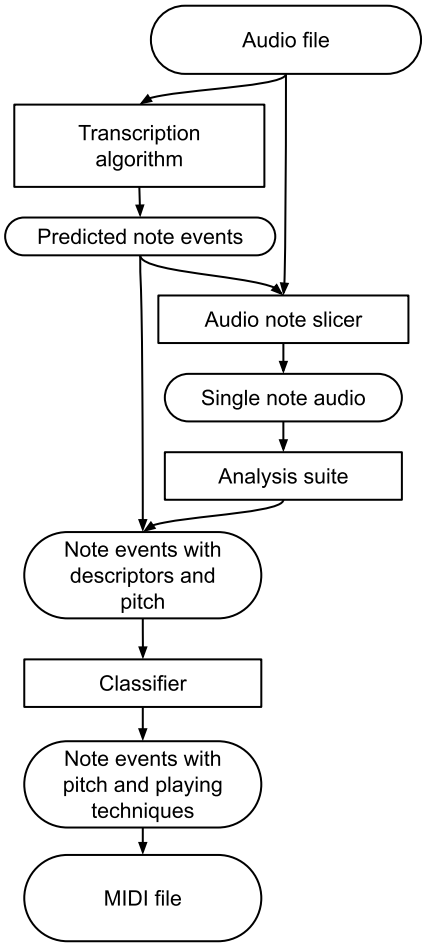


Figure 6-1: System implementation overview

6.1.2 Note segmentation

For note segmentation, we use Essentia's PitchContourSegmentation extractor[2], which implements the algorithm from McNab et al. ([19]). It uses the audio and the f0 estimates to generate discrete *note events*: objects with onset time, offset time, and pitch. These *note events* match how notes are annotated in the dataset, and are the level at which all further processing operates: the problem becomes finding the articulation of a given note event given the corresponding segment of audio and pitch.

6.2 Feature Extraction

After segmenting the audio into note events, we extract a large number of features for each note to be used as input for the classifier. For spectral features, we use a collection of Essentia's algorithms. We also implement additional features which we believe will help with detecting specific articulations. We extract each feature on the entire note's audio, and also on only the first 1024 samples of each note[3].

6.2.1 Timbral features

Similarly as in [8], we use a large number of timbral features that will then be fed into a feature selection algorithm before being used for classification. We trained classifiers using the features in Table 6.1, and compared excluding unproccessed array features, such as the MFCC, in favor of derived features like those in Essentia's Music Extractor

[2] `https://essentia.upf.edu/reference/std_PitchContourSegmentation.html`

[3] We compared extracting features from only the start of each note's audio, from the entire note's audio, or both. We found using both the features the first 1024 samples and from the entire note's audio length yielded the best performance.

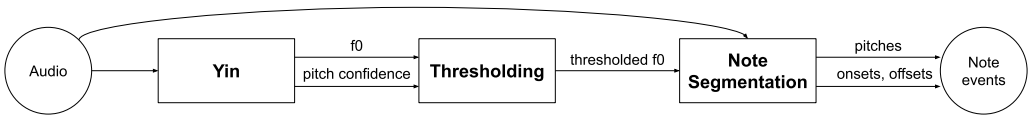


Figure 6-2: Pitch detection overview

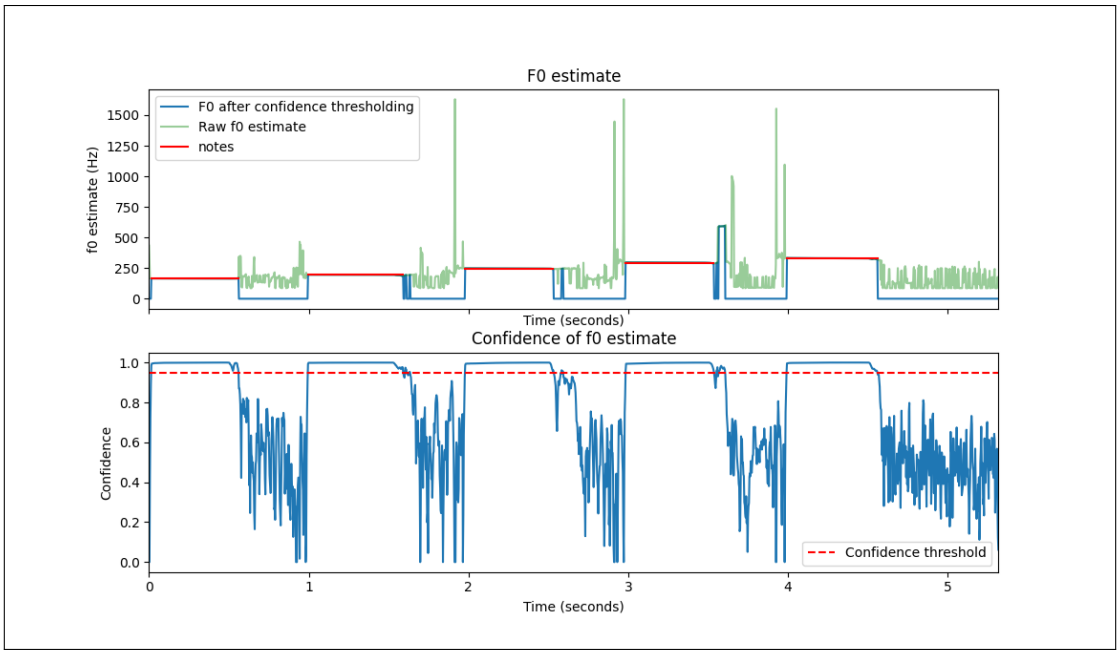


Figure 6-3: Transcription for `SF Lick 1 pick.wav`

(see Table 6.1). We found the array features to improve accuracy and included them in the final model.

6.2.2 Pitch features

Vibrato and slide are feature that can be detected from pitch alone. To improve accuracy for those articulations, we added the following features.

Slide detection using f0 derivative

In order to detect sliding up to a note, we analyse the pitch trajectory at the start of each note. We take the discrete derivative of the f0 trajectory at the start of the note using `numpy.gradient`[4], and use the maximum, minimum mean, and median values of the derivative as features. A slide should have a high maximum value and mean, whereas a static note would have a near zero derivative, and a vibrato would result in a near-zero mean since the derivative would switch signs frequently, oscillating fairly symmetrically around 0.

[4] `https://numpy.org/doc/stable/reference/generated/numpy.gradient.html`

Vibrato Detection

Vibrato is the slow and small modulation of the frequency of a note. We use the thresholded f0 values as input for essentia's Vibrato detection algorithm[5] in order to get vibrato extent and frequency (i.e. the width and speed of the frequency modulation) per frame for each given note. For each note, we then extract the maximum, median and mean values for both vectors, and use these values as features for the classifiers. Since we are simply detecting the presence or absence of vibrato, we are not concerned with the minimum value.

6.2.3 Loudness

We determine loudness in order to convey dynamics in the generated MIDI file. To do so, we used Essentia's Loudness algorithm[6]. We convert loudness to MIDI velocity using the following formula: $velocity = (1 - e^{-loudness/2.5}) * 126 + 1$ (see velocity curve on Figure 6-4).

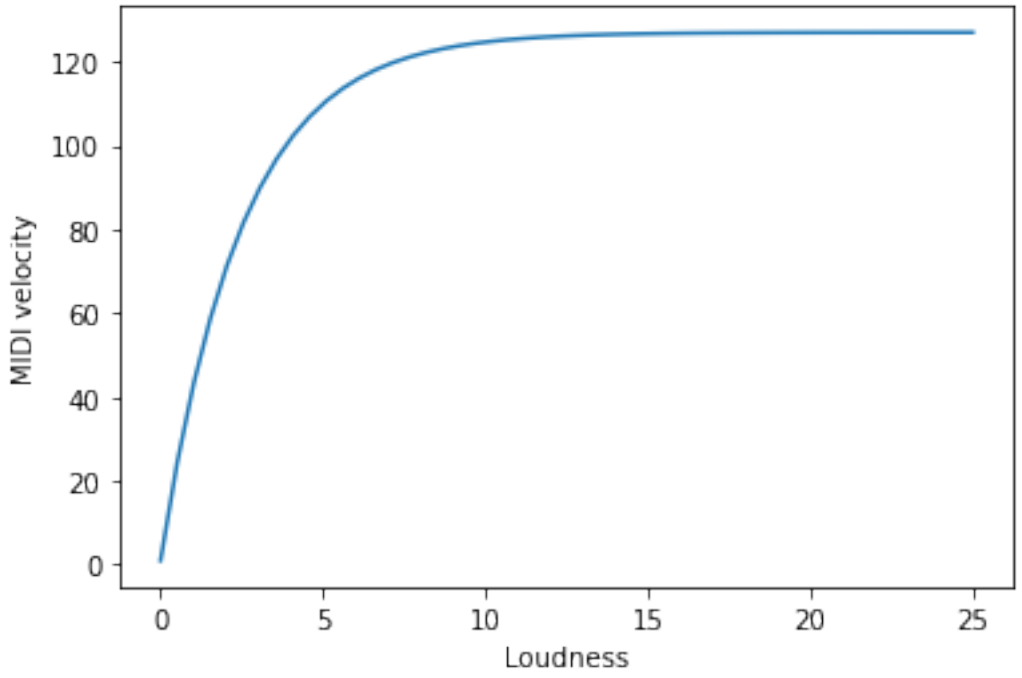


Figure 6-4: Loudness to MIDI velocity curve

[5] https://essentia.upf.edu/reference/std_Vibrato.html
[6] https://essentia.upf.edu/reference/streaming_Loudness.html

6.3 Classification

The features we extracted in the previous phase are used as an input to two classifiers: one for excitation style (fingerstyle, palm-mute, pick) and one for expression style (normal, vibrato, slide, bend, dead-note, harmonic). They were both built using the `scikit-learn` Python library[20].

6.3.1 Model Architecture

Both models follow a similar structure: Data Normalization -> Feature Elimination -> Multiclass classification (6-5). After evaluating a number of different models, we find that the same architecture worked best for both excitation and expression styles.

Data normalization

We first scale our features to zero mean and unit variance. We used `scikit-learn`'s `StandardScaler`[7] implementation with default parameters. This same scaling is also applied during prediction.

Feature Elimination

For each classifier, we trained a random forest to determine the importance of each feature, and kept all features with above average importance. This dropped the number of features from 180 to 158 for both models. We used `scikit-learn`'s `SelectFromModel`[8] and `RandomForestClassifier`[9] implementation with default settings.

[7] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html`

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

Feature type	**Feature**
Array features	Bark Bands
	MFCC
	Linear Predictive Coefficients
Spectral descriptors	High Frequency Content
	Spectral Contrast
	Harmonic Peaks
	Harmonic Magnitudes
	Inharmonicity
SFX descriptors	Envelope max to total, min to total, temporal centroid to total
	Pitch Salience
Music Extractor	Loudness
	Dynamic Complexity
	Spectral RMS
	Spectral Flux
	Spectral Centroid
	Spectral Moments
	Spectral Rolloff
	Spectral Decrease
	Spectral Strongpeak
	Spectral Energy
	Zero Crossing Rate
	Low, Low-mid, High-mid, High Frequency Spectral Energy
	MFCC Crest
	MFCC Flatness
	MFCC Centroid
	MFCC Moments
	MFCC Spread
	Bark Bands Crest
	Bark Bands Flatness
	Bark Bands Centroid
	Bark Bands Moments
	Bark Band Spread
	Spectral Entropy
	Spectral Complexity
Pitch Features	Vibrato Extent Max, Median, Mean
	Vibrato Frequency Max, Median, Mean
	F0 Gradient Max, Min, Median, Mean

Table 6.1: List of features

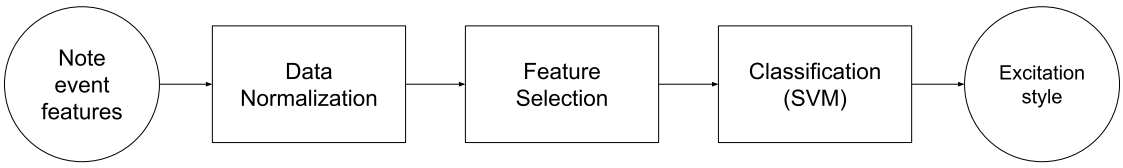


Figure 6-5: Classification Pipeline

Classification

We use a Random Forest trained on the scaled and pruned features. Once again, we use `scikit-learn`'s implementation[10]. After running a grid search, we settled on the following parameters: `max_features='sqrt', class_weight='balanced'`.

[10] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

Chapter 7

Evaluation

7.1 Transcription

To evaluate the pitch detection and note segmentation algorithm, we use the SMT-IDMT and the Arty datasets. The datasets contain the ground truth with regards to note event pitch, onset time, and offset time. To compare the prediction with the dataset, we match note events that have sufficiently close onsets, and record the differences in onset time, offset time, and pitch. We also count false positives (extraneous note events that were predicted but that don't correspond to real notes) and false negatives (notes played that weren't transcribed into note events), and used them to compute recall, precision, and F-score (Table 7.1).

With these criteria, we compare two of Essentia's pitch estimators: Yin [1] and Melodia [2], implementing algorithms from DeCheveigné and Kawahara ([18]) and Salomon and Gomez ([21]) respectively. Following the results in Table 7.1, we use Yin with a confidence threshold of 0.95, which yields an F-score of 0.79 on our dataset.

[1] `https://essentia.upf.edu/reference/std_PitchYin.html`
[2] `https://essentia.upf.edu/reference/std_PredominantPitchMelodia.html`

Pitch algorithm	Confidence Threshold	Precision	Recall	F score
Yin	None	0.38	0.94	0.54
	0.95	0.66	0.99	0.79
Melodia	None	0.60	0.95	0.73

Table 7.1: Pitch detection scores

7.2 Classification

Models are evaluated using 5-fold cross-validation on the IDMT and Arty datasets. Several model architectures are compared (see Tables 7.2 and 7.3). For all the models considered, the parameters for the most promising architectures were selected using Grid searches. The best performing model, a random forest, has an overall accuracy is 0.879 for excitation styles (see Table 7.2) and 0.873 for expression styles (see Table 7.3) on the Arty dataset. On the IDMT dataset, we improve on Kehling et al.'s ([1]) accuracy with scores of 0.93 and 0.89 for excitation and expression styles respectively (see Table 7.4). Per-articulation performance on the Arty dataset can be seen in Table 7.5 and on the confusion matrices (Figure 7-1)[3].

7.2.1 Discussion

Tables 7.2 and 7.3 show that Random forests are the most effective classifiers for this task, achieving highest accuracy and F-scores during cross validation on both the IDMT and Arty datasets . When trained and evaluated on the IDMT dataset alone, they performs better than Kehling et al. [1] (see Table 7.4). However, they also achieve perfect accuracy on the training dataset while performing significantly worse on the test dataset, suggesting over-fitting. After getting the preliminary results from the first user study, we selected another model with a smaller discrepancy between training and test scores to try to reduce over-fitting for the second user study. Using Support Vector Machines seems to lead to less overfitting, and using Random Forest

[3]There are too few bends and dead-notes in the dataset to get meaningful accuracy figures, but we keep them as a category to allow for comparison with Kelhing et al.

Feature Selection Algorithm	Classifier	Accuracy		F-score		Recall		Precision	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
Random Forest[a]	SVC	0.832	0.011	0.842	0.010	0.855	0.011	0.839	0.012
None	SVC	0.817	0.010	0.826	0.010	0.835	0.012	0.823	0.010
Recursive Feature Elimination on a Random Forest	SVC	0.847	0.017	0.856	0.012	**0.864**	0.012	0.851	0.013
Sequential Feature Selector	SVC	0.800	0.003	0.810	0.014	0.822	0.014	0.808	0.013
Random Forest[b]	Random Forest	**0.879**	0.014	**0.870**	0.014	0.861	0.012	**0.883**	0.017
None	Random Forest	0.848	0.018	0.849	0.018	0.838	0.175	0.869	0.016

Table 7.2: Performance of excitation style models on full dataset

[a]Used for second round of user study
[b]Used for first round of user study

Feature Selection Algorithm	Classifier	Accuracy		F-score		Recall		Precision	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD
Random Forest[a]	SVC	0.840	0.018	0.651	0.029	**0.701**	0.040	0.631	0.024
None	SVC	0.866	0.021	0.674	0.032	0.629	0.029	**0.794**	0.081
Sequential Feature Selector	SVC	0.843	0.032	0.650	0.053	0.609	0.055	0.763	0.0697
Random Forest[b]	Random Forest	**0.873**	0.020	**0.675**	0.065	0.639	0.054	0.751	0.103
None	Random Forest	0.871	0.020	0.666	0.045	0.631	0.052	0.727	0.041

Table 7.3: Performance of expression style models on full dataset

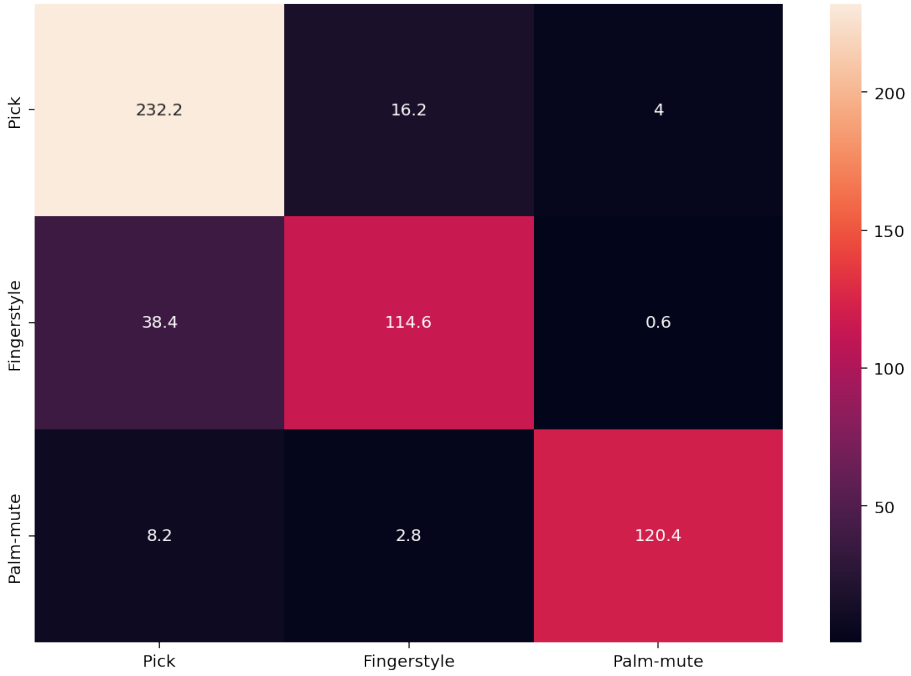
[a]Used for second round of user study
[b]Used for first round of user study

as a means of feature selection still proved to be a better strategy over alternatives like Recursive Feature Elimination or Sequential Feature Selection.

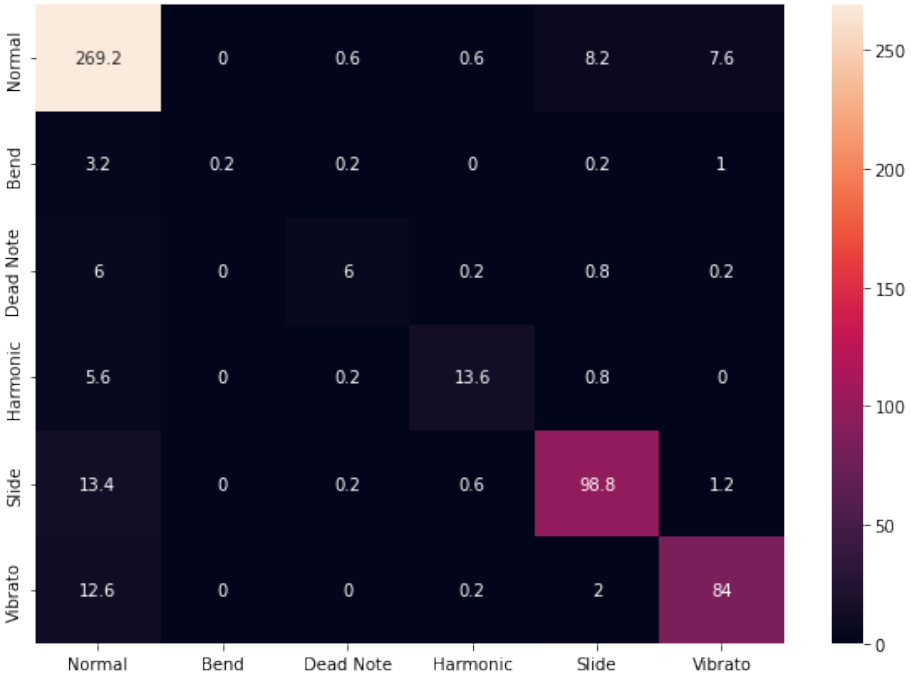
In the excitation style confusion matrix (Figure 7-1), we can see that most of the errors happen between fingerstyle and pick. Some degree of confusion is to be expected between playing fingerstyle and with a pick, as differentiating them can be a hard task for even a human listener. Even though on a given instrument, playing fingerstyle will usually result in a "rounder" tone, with a less pronounced attack and less high frequencies, the differences between different electric guitars make the classification more difficult if notes are heard out of context.

7.3 User Study

We conducted two rounds of user studies: a first preliminary round with 2 user, and a wider round with 9 users. We adjusted the model between rounds based on the feedback from round 1, so the evaluation will mainly discuss data from round 2. We



(a) Excitation style



(b) Expression style

Figure 7-1: Confusion matrices

Model	Excitation	Expression
Arty	0.93	0.89
Kehling et al.	0.93	0.82

Table 7.4: Performance on IDMT dataset compared to Kehling et al. [1].

	Pick	Fingerstyle	Palm-mute
Accuracy	0.920	0.746	0.916

(a) Excitation styles

	Normal	Harmonic	Slide	Vibrato
Accuracy	0.941	0.673	0.865	0.850

(b) Expression styles

Table 7.5: Arty cross-validation accuracy per playing technique.

recruited users with experience using virtual instruments and playing guitar, from the "Virtual Orchestration" Facebook group[4]. They overwhelmingly were advanced DAW and virtual instrument users (see Tables 7.6 and 7-2). They were also experienced guitarists, playing for a median of 15 years, and a minimum of 4 (see Figure 7-3). They used a variety of electric guitar models, with both single coil and humbucker guitar pickups being represented. They reported playing a variety of musical genres, most commonly rock (7 mentions) and jazz (5 mentions).

The user study was composed of two tasks: the first task provides us with out-of-dataset samples to evaluate Arty, while the second task encourages users to use Arty more freely.

[4] `https://www.facebook.com/groups/1475801049334232`

Users DAW experience	1	2	3	4	5	Mean	SD
Response Count	0	0	1	1	9	4.75	0.62

Table 7.6: Self reported DAW experience of Arty testers, on a scale from "never used a DAW before" (1) to "I produce music in DAWs regularly" (5).

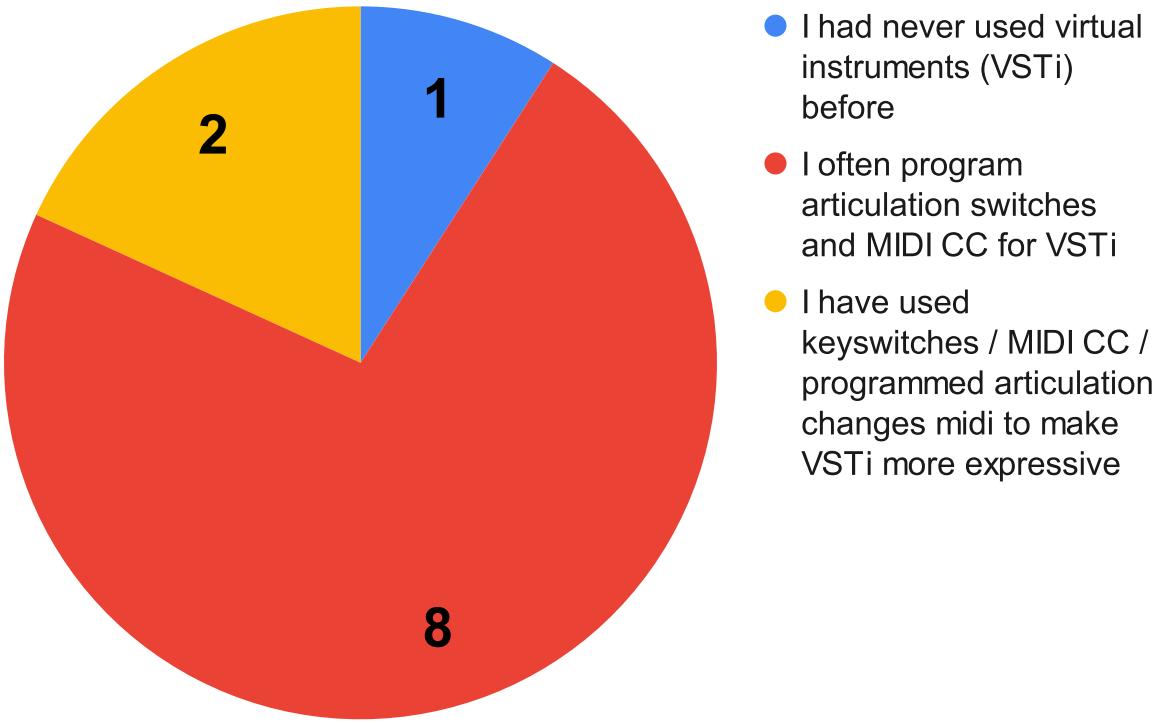


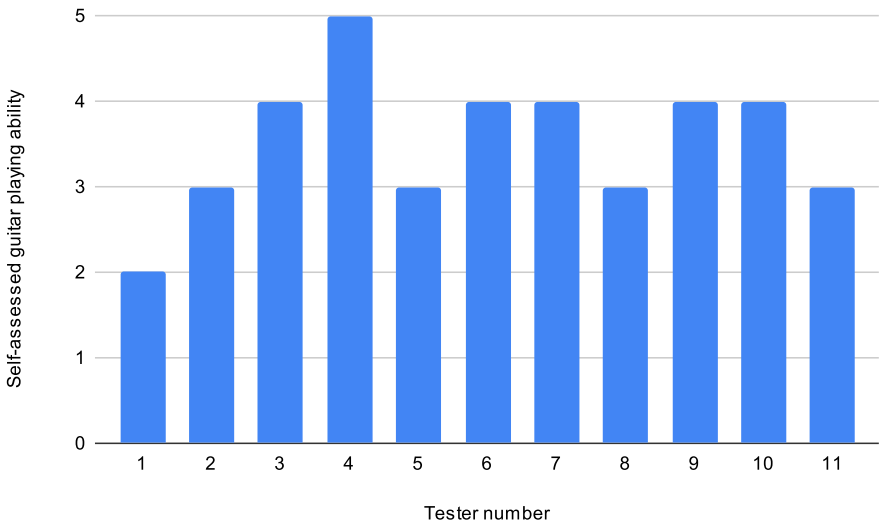
Figure 7-2: User virtual instrument experience

7.3.1 Task 1

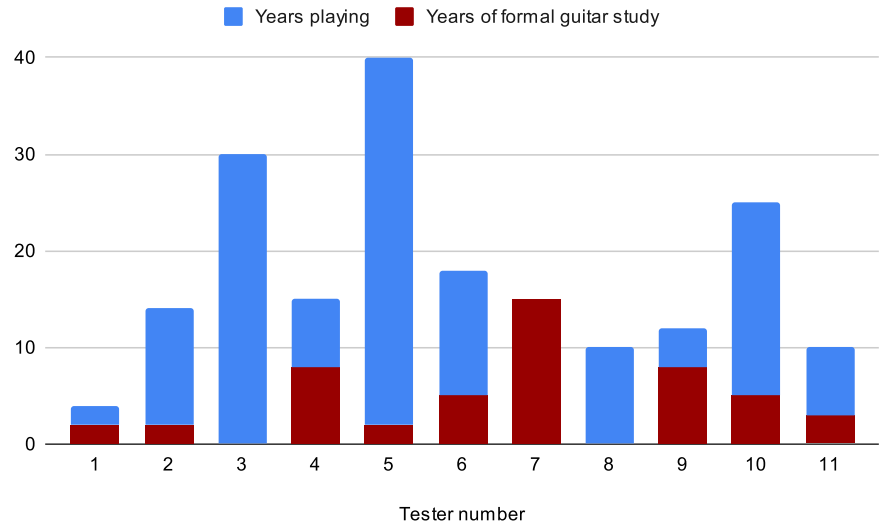
Task Description

The purpose of this task was to familiarize the users with Arty as well as collect some out-of-dataset samples that we could evaluate the model on. The task consisted of recording all the relevant techniques into Arty to assess whether Arty is accurate on data that resembles the dataset. The users were asked to play each articulation three times, on three different strings, following the sheet music in Figure 7-4. They then had to upload both the audio file of their playing as well as the generated MIDI file to a provided Dropbox[5] link. We requested that they use the neck pickup of

[5] `https://www.dropbox.com/`



(a) Self-assessed guitar proficiency on a scale from 1 (beginner) to 5 (professional level)



(b) Guitar playing experience in years

Figure 7-3: Tester guitar playing experience

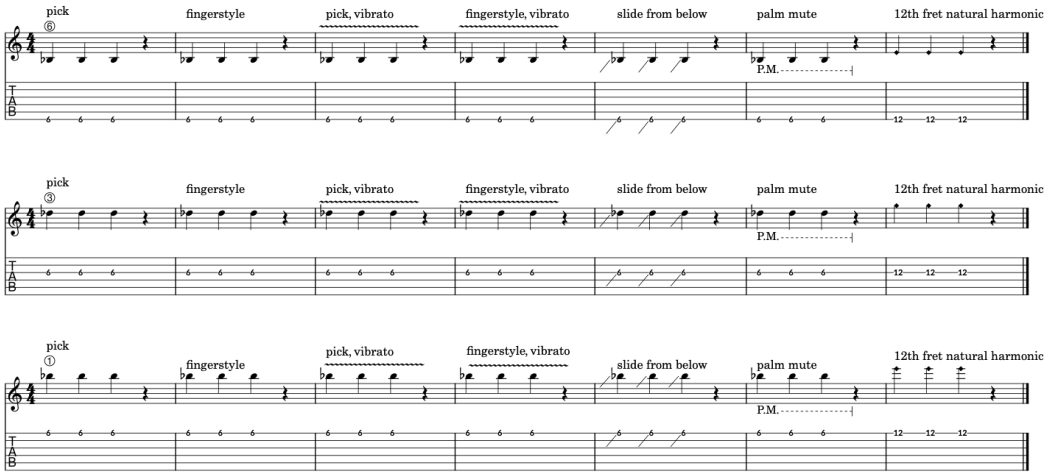


Figure 7-4: Task 1 sheet music

their guitar, as the Arty dataset was recorded using the neck pickup, and imposing a specific pickup does not impede on Arty's design goals, as it does not significantly change how guitar is played.

Results and discussion

Comparing the audio and MIDI files sent by users, we were able to evaluate Arty's performance for different users for the first task. Of the 11 users, one did not send the correct audio, and one didn't follow the sheet music perfectly and skipped a palm-muted note. All other data is included in the following discussion.

Pitch tracking In the first round of user study, we observed a high rate of false positives and false negatives (10 and 17 respectively, out of a total of 126 notes played by the users total). Harmonics in particular were not being transcribed. We adjusted the transcription algorithm's parameters and threshold before round 2, which improved the performance significantly (see Table 7.7). The following transcription discussion will be restricted to round 2 results, using the adjusted algorithm.

The pitch algorithm performed well overall, but had particular trouble with wide vibrato. Vibrato, especially wide vibrato, sometimes resulted in notes with neighboring pitches being added, since the pitch approaches the semi-tone above. 17 such

User Study Round	Precision	Recall	F score
Round 1	0.916	0.865	0.890
Round 2	0.931	0.990	0.960

Table 7.7: Transcription performance for task 1

	Playing technique used when the error occurred					
Transcription metric	Vibrato	Slide	Harmonics	Palm mute	Other	Total
False positive count	17	9	6	0	5	37
False negative count	0	0	3	2	0	5
True positive count	72	72	69	69	216	498

Table 7.8: Transcription performance per articulation in user study round 2, task 1.

false positives were found in round 2 (see Table 7.8), with 10 being from a single user with a particularly wide vibrato. Incorporating feedback from vibrato detection to tune the pitch detection algorithm and prevent extraneous notes from being added could improve this behavior.

Similarly, sliding up sometimes led to additional MIDI notes preceding the target pitch, a semitone lower (8 total in round 2). It also had trouble with harmonics. Harmonics tend to be quieter than regular notes while still having a noisy attack due to the pluck, which results in a more percussive sounds and lower pitch confidence. If the pitch confidence drops low enough, the thresholding will prevent the note from being recognized. This issue with harmonics depends on the player and is especially pronounced for less experienced guitarists: all the harmonics that were not transcribed were from the same user.

Playing technique detection Arty exhibits low performance on playing technique detection in task 1, always predicting fingerstyle vibrato (59% of results, see Table 7.9), pick vibrato (24% of results), or palm mute (17% of results). It never detects absence of vibrato, but even disregarding vibrato, accuracy is still low at 43% (see Table 7.10). Performance differs greatly between users: some users saw most of their notes translated as "sustain" (up to 62% of notes for one user), while another user

	Fingerstyle Vibrato	Pick Vibrato	Palm Mute	Total
Count	296	118	84	498
Proportion	59%	24%	17%	100%

Table 7.9: Playing techniques detected on user study round 2, task 1

Target articulation		Marcato	Sustain	Marcato vibrato	Sustain vibrato	Tremolo	Pizzicato	Total
	Predicted	53	26	0	0	0	54	133
	Accuracy	0.74	0.36	0	0	0	0.75	0.26
Ignoring vibrato	Predicted	109	53	N/A	N/A	0	54	216
	Accuracy	0.76	0.37	N/A	N/A	0	0.75	0.43

Table 7.10: Playing technique detection performance for user study round 2, task 1

only had 1 note classified as such by Arty.

This suggests that differences between users are greater than differences between a user using different playing techniques, with the exception of palm-mute. Pronounced palm-mute (resulting in shorter notes) was consistently detected correctly, while notes decaying more slowly due to a lighter palm mute were sometimes miss-classified.

Overall, Arty mostly failed to generalize from its training dataset to real data, limiting itself to the most common categories, and classifying differences between players and guitars rather than between playing techniques.

7.3.2 Task 2

Task Description

The purpose of task 2 was to let the users explore Arty and evaluate its playing technique detection in a less restrained setting. Users were instructed to record melodies of their choice into Arty, using all the recognized playing technique. They were then asked to play the generated MIDI file in a DAW of their choice, using and the "Strings - Violin Solo 2 KS.sfz" soundfont[6] in Plogue Sforzando[7]. They then evaluated Arty's accuracy and user experience. The exact instructions were:

[6] `https://github.com/peastman/sso`

[7] `https://www.plogue.com/products/sforzando.html`

Arty listens to your guitar audio, transcribes the notes and playing techniques, and makes a MIDI file that will reflects them. When the virtual violin plays that MIDI file, it will follow the notes and playing techniques you used. The map between guitar and violin techniques is as below:

- Right hand techniques:
  - Palm-mute -> Pizzicato
  - Fingerstyle -> Marcato
  - Playing with pick -> Sustain
- Left hand techniques:
  - Guitar harmonics -> Tremolo
  - Slide up from below -> Tremolo
  - Vibrato -> Vibrato (for sustain and marcato)

Notes:

- Playing palm mute will always result in pizzicato, regardless of left hand technique.
- Vibrato can apply to marcato and sustain articulations (but not pizzicato)
- Tremolo is the same, regardless of playing fingerstyle or with a pick

Results and discussion

Pitch tracking We asked the users to rate the pitch tracking and to identify specific issues with it. The pitch tracking was rated rather highly (see Table 7.11), with only one user reporting octave errors, and no other pitch errors being reported.

Three users reported timing errors, but we couldn't reproduce two of them. One

noticed a delay between the audio and the generated MIDI, but we found no such delay when looking at the audio and MIDI files collected in task 1. Another user reported the overall tempo of the MIDI to be slower than the audio's tempo, a behavior which we could also not reproduce with the files they sent. We believe the reported differences in tempo could be due to a non-standard tick-per-quarter setting in the users' DAWs, which would make the MIDI playback at a different rate than intended.

The last timing issue reported was about note cutoffs, particularly regarding palm mute and sustained notes. Such cutoffs are hard to determine objectively because, unlike a violin which can sustain notes at a constant volume until the bow stops exciting the string, notes played on a guitar decay progressively into silence. Our approach while authoring the Arty dataset was to set the cutoff where the note was no longer audible to us. This is not a precisely defined point, and probably lead to inconsistencies in the dataset. These in turn might explain this user's issue.

Although this shouldn't be too much of an issue when using Arty with a violin VSTi, because palm mute maps to pizzicato (which similarly decays regardless of MIDI note length) and other situations usually call for a clear cutoff, it could be an issue if trying to use Arty to control MIDI plucked string instruments. Modifying the dataset to have a consistent cutoff point, determined for example by a signal-to-noise threshold, or a loudness ratio compared to the loudest point in the note, might improve Arty's cutoff behavior.

The most reported issue with the pitch tracking was about note segmentation, with 3 users reporting notes not being transcribed and 4 users reporting extraneous notes being added. This happened in particular with harmonics, palm mute, and vibrato. Harmonics and vibrato are challenging for Arty for reasons described above, and similarly to harmonics, palm mute results in a strong percussive attack and a quieter sustained pitch.

In open comments, testers were overall satisfied with the pitch tracking, but noted DAWs already offer accurate tools for pitch detection, such as ReaPitch in Cockos Reaper[8]. These tools also often work in real-time, making them more convenient to

[8] `https://www.reaper.fm/index.php`

Pitch tracking user rating	1	2	3	4	5	Mean	SD
Response count	0	0	2	5	4	4.18	0.75

Table 7.11: Pitch tracking user rating, on a scale from 1 to 5 (best)

use if only pitch is concerned.

Articulation detection Confirming the results from the first task, articulation detection was rated poorly by users, with only fingerstyle, pick, and palm mute being reliably detected for a few users (see Table 7.12). When asked whether testers would use Arty in their regular music making, this low accuracy was the main component dissuading testers, being mentioned by 8 testers.

User Experience The second biggest deterrent to using Arty was DAW integration, with 6 mentions. Users found that leaving their DAW for a browser was disruptive to their workflow, and would prefer an implementation as a VST plugin. Arty was made as a web application to facilitate development and user studies, but a product designed to be used by professionals would most likely need to be implemented as a VST plugin to work offline and within a DAW.

Three users put real-time feedback or transcription as a requirement. Near real-time pitch detection would be possible, although note segmentation would be more difficult and likely less accurate. However, real-time playing technique detection would be much more difficult. A classifier trained on only the start of the notes could give reasonable feedback with a bit of lag, but true real-time is still out of reach for this kind of architecture.

Overall, users were very enthusiastic towards using a hypothetically more accurate and integrated version of Arty, with only one user reporting they wouldn't use Arty even if these issues were fixed. The other users found an improved Arty to be an improvement on their current input methods: using piano keys or special buttons on a hardware controller to trigger keyswitches (4 users), or inputting them with a mouse and keyboard after entering the notes (3 users). 5 users reported having to

Articulation trigger reliability	Pick	Pick, vibrato	Fingerstyle	Fingerstyle, vibrato	Palm Mute	Harmonics	Slide
I didn't play that articulation	0	0	0	1	0	1	3
Never triggers	1	5	2	5	1	8	6
Sometimes Triggers	2	1	5	2	2	1	1
Rarely Triggers	3	5	1	2	1	1	1
Sometimes Triggers	2	1	5	2	2	1	1
Often triggers	3	0	2	1	5	0	0
Reliably Triggers	2	0	1	0	2	0	0

Table 7.12: User ratings of perceived classification accuracy per articulation

tweak the expressive data even if they start with live inputting notes with a keyboard. A more accurate Arty would limit the need to do this second pass, while leaving the possibility open. However, no user was interested in using Arty as is.

There were a few issues brought up about what playing techniques Arty should detect. We chose the guitar paying technique to articulation mapping based on what articulation were available in our dataset to train the model, and what articulations we could find on a free virtual instrument that could be easily installed by testers. We didn't use bending as a possible articulation as it would have mapped to pitch variations rather than articulation switches, but two users would have liked for bends to be recognized. Two users also mentioned that they would have wanted tremolo picking on guitar to be recognized.

Two users were dissatisfied with the guitar-playing-technique-to-violin-articulation mapping, because they found it to be counter-intuitive and/or missing some common articulations such as *Spiccato*. Because of the limited articulations available in free VSTis, we had to assign some counter-intuitive technique to articulation mappings such as guitar harmonics mapping to violin tremolo (because violin harmonics were not an available articulation). This limitation would be overcome by offering user-customizable mappings, or by targeting paid products with a wider range of articulations.

One user also reported being unhappy with the guitar to MIDI velocity mapping. MIDI controllers often offer several velocity curves to adjust for player preferences, and this could be added as a user-controllable parameter to Arty.

Chapter 8

Conclusion

Arty is an attempt to combine the expressivity and spontaneity of end-to-end timbre transfer with the compatibility and tweakability of MIDI, by transcribing guitar playing techniques and using that information to pilot third party virtual instruments. Building on previous work by Kehling et al. [1], we extended their dataset with the new Arty dataset, and used it to train a model to classify excitation styles and expression styles. We implemented a website as a user interface to allow users to easily convert their guitar playing to MIDI. Despite achieving fairly high accuracy on the dataset, the user study showed that Arty's high cross-validation score didn't convert to a high real-world accuracy. Potentials reasons for this shortcomings and next steps are discussed below.

8.1 Future Work

8.1.1 Classification accuracy

Although fairly accurate on the IDMT and Arty datasets, Arty's performance during the user study shows the articulation detection needs to be greatly improved before it is ready to be used in a musical context. Providing the model with more diverse data from a variety of players and electric guitar models would increase the model's robustness, but would be difficult and time consuming to assemble and annotate.

We attempted this approach by creating the Arty dataset to complement the IDMT dataset, but the amount of additional data needed is larger than we anticipated. This is a problem that would most likely not be solved by training a different model on existing data, since Arty's current model can achieve relatively high accuracy on our current data. This is a bottleneck for any approach based on a trained classifier. Although there are several such attempts in the literature, there is no agreed-upon set of playing techniques to detect, which makes combining datasets and comparing results difficult.

An alternate approach that might be able to succeed without requiring too much data would be to focus on a hand-tuned classifier, by "manually" setting thresholds for a small number of hand picked features for example. Although this approach limits the complexity of the classification model, and hence might make it more difficult to achieve high accuracy on the dataset, it is easy to adjust, opening the door to user calibration. We observed in the user study that timbral differences between players and guitars were sometimes larger than differences between playing techniques, especially playing fingerstyle vs using a pick. By adding a user calibration stage resembling task 1 of the user study, a simpler model could be adjusted to account for these differences between guitars and potentially exhibit better performance on out-of-dataset samples.

8.1.2 User experience

According to user feedback, beyond accuracy, improving the user experience is largely a matter of offering a DAW integrated version, and real-time feedback.

A DAW integrated version of Arty would most likely be a VST plugin that could be inserted on any audio track, and could output the corresponding MIDI. Usual VST plugins have to work in near real-time, with the possibility of having a small delay. This wouldn't work for Arty's current non-real-time design, however using the ARA[1] SDK, an Arty plugin could access the entire audio and process it without a real-time constraint while still being cross-DAW compatible.

[1] `https://github.com/Celemony/ARA_SDK`

Real-time playing technique classification would need to rely only on the start of each note, and this would prove a more difficult task than being able to use the whole note. For certain techniques where the attack of the note is significantly different (palm mute, slide for example) this might be possible. However, it would make certain playing techniques impossible to detect, such as vibrato.

A compromise could be to output the pitch in real time to give some real-time feedback, and perhaps even the detected playing technique after the note is over. Arty could then add the keyswitches and MIDI CC after the fact into a MIDI file that would be dragged and dropped onto the track. Although not fully real-time, this might give enough feedback to the user while remaining feasible with Arty's classification architecture.

8.1.3 Compatibility

Currently, because key-switches have to be tailored to a particular virtual instrument, Arty only works with a single instrument. Expanding Arty's compatibility would be a fairly simple task, either by adding a lists of mappings for popular virtual instruments, or better yet by enabling users to create their own mappings from playing technique to keyswitches. Providing presets for popular instruments would help keep Arty intuitive. Other parameters that could be given to the user are an adjustable guitar loudness to MIDI velocity curve, as is often the case in hardware MIDI controllers.

Appendix A

Arty Dataset description

A.1 Intention

The Arty dataset is intended as a ground truth for the development of music information retrieval algorithms. It consists of annotated audio recordings of the author playing a variety of playing techniques on electric guitar. It was designed for the following intended applications:

- monophonic pitch estimation
- onset/offset detection
- guitar playing technique detection

It follows the same format as dataset 2 from the IDMT-SMT-GUITAR dataset [1], and is meant to be used in conjunctions with it.

A.2 Annotations

The following information is annotated for every note:

- pitch (midi number)
- onset time (in seconds)

Abbreviation	Excitation Style
FS	Finger-style
MU	Palm-muted
PK	Picked

Table A.1: List of excitation styles

Abbreviation	Expression Style
DN	Dead note
HA	Harmonic
NO	No expression style
SL	Slide (pitch)
VI	Vibrato

Table A.2: List of expression styles

- offset time (in seconds)
- plucking style
- expression style

Excitation and expression styles correspond to right hand and left hand playing techniques respectively. A list of styles is presented in tables A.1 and A.2.

A.3 Dataset Statistics

A.4 Content

The Arty dataset consists of audio files and annotation files

A.4.1 Audio files

The audio files were recorded using an Ibanez JSM-10 electric guitar plugged directly into an M-audio 2x2M or Behringer UM-2 audio interface. The neck pickup was used

Playing Technique		SMT-IDMT-GUITAR-V2 dataset	Arty dataset	Combined dataset
Excitation Style	Pick	779	221	1000
	Fingerstyle	127	216	343
	Palm Mute	342	138	480
Expression Style	Normal	778	251	1029
	Bend	28	0	28
	Dead Note	237	0	237
	Harmonic	103	0	103
	Slide	33	180	213
	Vibrato	69	144	213

Table A.3: Playing technique occurrences

	PK	FS	MU
NO	395	101	282
BE	12	7	9
DN	219	9	9
HA	92	2	9
SL	21	0	12
VI	40	8	21

(a) IDMT-SMT-GUITAR_V2

	PK	FS	MU
NO	89	84	78
BE	0	0	0
DN	0	0	0
HA	0	0	0
SL	60	60	60
VI	72	72	0

(b) Arty dataset

	PK	FS	MU
NO	484	185	360
BE	12	7	9
DN	219	9	9
HA	92	2	9
SL	81	60	72
VI	112	80	21

(c) Combined Datasets

Table A.4: Occurrences of expression/excitation style pairs

for all recordings. The audio was recorded into Logic Pro X at 44100Hz and exported as wav files.

A.4.2 XML annotation files

The annotation files are `.xml` files similar to those used in the IDMT-SMT-Guitar dataset 2. They have the following strucure:

```
<?xml version='1.0' encoding='UTF-8'?>
<instrumentRecording>
  <globalParameter>
    <audioFileName>SF Full Chromatic pick SL.wav</audioFileName>
    <instrument>EGUI</instrument>
    <instrumentModel>Ibanez JSM10</instrumentModel>
    <pickupSetting>Neck</pickupSetting>
    <instrumentTuning>40  45  50  55  59  64</instrumentTuning>
```

```
    <recordingDate>03/18/2022</recordingDate>
    <recordingArtist>Sebastian Franjou</recordingArtist>
    <instrumentBodyMaterial>Maple</instrumentBodyMaterial>
    <instrumentStringMaterial>Steel</instrumentStringMaterial>
    <composer>Sebastian Franjou</composer>
  </globalParameter>
  <transcription>
    <event>
      <onsetSec>1.793015873</onsetSec>
      <pitch>43</pitch>
      <offestSec>2.5396825400000003</offestSec>
      <excitationStyle>PK</excitationStyle>
      <expressionStyle>SL</expressionStyle>
      <fretNumber>NA</fretNumber>
      <stringNumber>NA</stringNumber>
      <modulationFrequency>NA</modulationFrequency>
      <modulationFrequencyRange>NA</modulationFrequencyRange>
    </event>
    <event>
        ...
```

Listing A.1: dataset XML format

The following tables describe each parameter and are taken from the IDMT-SMT-Guitar dataset description:

A.4.3 Other annotation files

We also provide other annotation files that correspond to intermediate stages in the creation of the dataset (see figure A-1). These could be useful if further annotation is needed. These files can be used to recreate the final dataset using functions in `dataset.py`.

Parameter	Description	Mandatory
audioFileName	file name of corresponding audio file	yes
Instrument	instrument name, see table below	yes
instrumentModel	e.g. "Fender Stratocaster"	no
pickUpSetting	pick-up combination used	no
instrumentTuning	tuning of all open strings given as MIDI pitch values, e.g. 28,33,38,43 (4 string bass guitar)	no
audioFX	audio effects used	no
recordingDate	date of recording	no
recordingArtist	person playing the instrument	no
instrumentBodyMaterial	instrument body material	no
instrumentStringMaterial	instrument string material	no
Composer	composer of recorded melody / music piece	no
recordingSource	source of recording	no

Table A.5: List of global parameters

Dorico file

The sheet music for the notes played is stored in `.dorico` format, corresponding to the music engraving software Steinberg Dorico 4. It is a single file with each *flow* (Dorico's equivalent to a musical movement) corresponding to a different audio file. The rhythm information was not used and should be discarded. however the expression and excitation styles are encoded as per table A.7 to ensure it would be exported correctly to `.musicxml`. In addition, it contains information about the string on which each note is played, which could be used to annotate string and fret information for each note.

MusicXML files

This is a `.musicxml` export of the sheet music described above. It contains all the information described above except for the string information. The name of the articulations can be seen in table A.7.

Sonic Visualizer Annotated layer

Each `.txt` file corresponds to an audio file. Each line has three tab-separated numbers: the onset time in seconds, the region number and the offset time in seconds.

Parameter	Description	Mandatory
Pitch	MIDI pitch value	yes
onsetSec	absolute note onset in seconds	yes
offsetSec	absolute note offset in seconds	yes
fretNumber	fret number where a note was played, starts with 0 for notes played on the open string	no, only relevant for guitar, bass guitar
stringNumber	string number where a note was played, starts with 1 for the lowest string	no, only relevant for guitar, bass guitar
excitationStyle	style which is used to excite the note (commonly referred to as plucking style for string instruments), see table below	no
expressionStyle	expression style which is used after note was excited, see table below	no
loudness	dynamic level expressed in "classical notation" (p, f, mf, ...)	no
modulationFrequencyRange	modulation frequency range in cent (e.g. quarter-tone bending 50 (cent)) no modulationFrequency (average) modulation frequency in Hz	no
(instrument)	in case notes from multiple instruments are annotated in one WAV file instrument must be set as note event parameter, if it is not set value of global parameter (instrument) is used	no

Table A.6: List of note event parameters

These were hand annotated from the audio files in sonic visualizer[1]. In the case of ringing notes, offset was determined to be when the note was no longer audible (even if some trace of the waveform remained upon visual inspection).

JSON files

The sonic visualizer and `.musicxml` files are combined to get a format with playing techniques, pitch and timing information. We call this the *note event* format. Internally, it is a python dictionary representing a single note with the following fields:

- onsetSec
- pitch
- offsetSec
- excitationStyle

[1] `https://www.sonicvisualiser.org/`

Playing Style	**Music notation**
Pick	Nothing
Fingerstyle	Tenuto
Palm mute	Staccato
Vibrato	Unstress
Dead note	X notehead
Slide (from below)	Scoop
Harmonic	Diamond Notehead

Table A.7: Playing style to articulation encoding

- expressionStyle
- vibratoExtent (optional)
- vibratoFreq (optional)

To each audio file corresponds a `.json` files which is a serialization of a list of note events, corresponding to the notes in the audio files as annotated by the author.

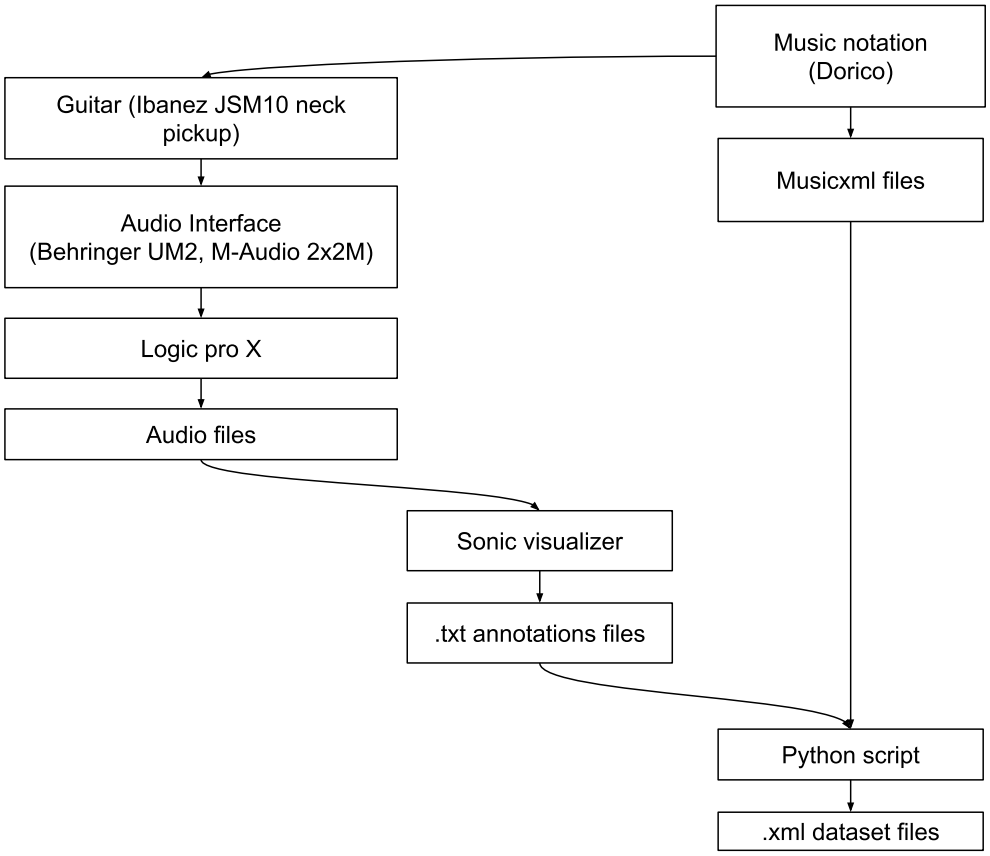


Figure A-1: Dataset pipeline

Appendix B

User Testing Form

Arty user testing form

Sign in to Google to save your progress. Learn more

* Required

Setup and instructions

Thanks for helping me in evaluating Arty, my Master's thesis project!

Arty is a website that turns your guitar into a violin. Arty listens to electric guitar audio. It detects not only the notes and rhythm, but also the playing techniques used. The violin sound will match the notes and techniques played, thanks to the keyswitches in the MIDI file. Arty only works with single notes (not chords), and requires a dry guitar signal recorded straight into an audio interface.

After completing this evaluation form, you will (if you wish) receive a 10$ amazon gift card. If you prefer to remain fully anonymous, just use a nickname of your choice made of 2 words and a number (such as guitarPlayer42)

Install instructions

STEP 0:

- download and install Plogue Sforzando from here: https://www.plogue.com/downloads.html#sforzando
- download "sso-small.zip" from https://www.dropbox.com/s/1a4w1sj716fexi4/sso-small.zip?dl=1 and decompress it (right click the file -> "Extract All" on windows, double click the file on Mac)
- download google chrome (if you don't have it already), Arty does NOT work on any other browser: https://www.google.com/chrome/

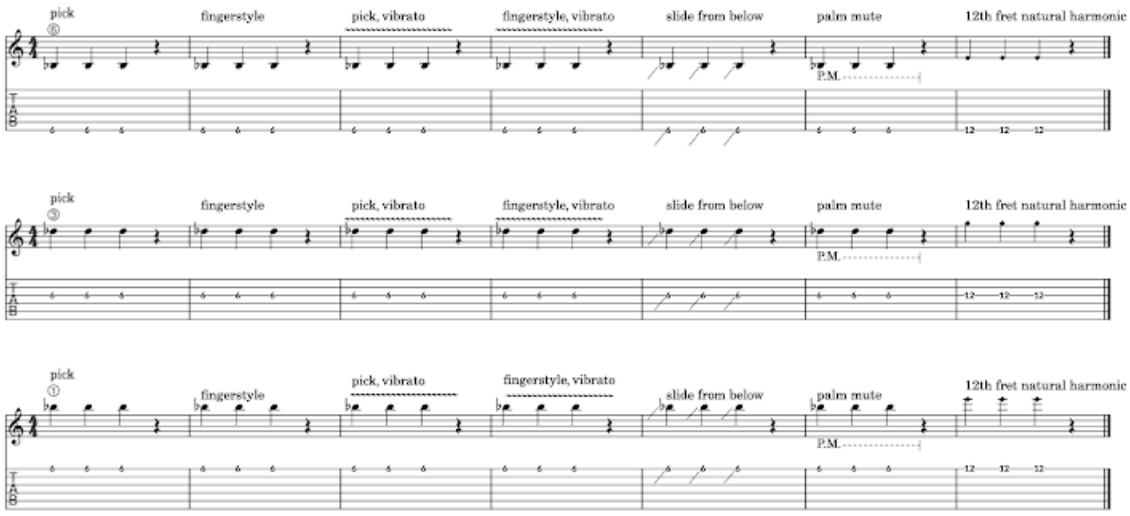
STEP 1:

- Visit arty at https://artymusic.azurewebsites.net/ using Chrome or Chromium
- Connect your guitar into your computer (using an audio interface)
- Make sure the system audio input is your music interface
  On Mac: System Preferences -> Sound -> Input -> select your audio interface
  On Windows: Settings -> System -> Sound -> Choose your input device
  Note: If done properly, the waveform display on the website should react to guitar playing but not to a clap. You might need to tweak the gain on your interface, plug into the first input of your interface, or refresh the page.
- Record a clip of yourself playing using the record button
- Click convert to download a MIDI version of your audio clip: the notes will match what you played, and there will also be extra MIDI information corresponding to the playing techniques you used

Alternatively, you can record yourself in your DAW of choice and upload the file to Arty using the "Choose File" button.

Tasks

Task 1: Record the following excerpt into arty, and convert it to MIDI. Use the neck pickup of your guitar if possible. Then upload both the audio and generated midi to this link, with your name (or chosen nickname) in the title: https://www.dropbox.com/request/iezjutzG1kGxL75z8okS *



- [ ] I uploaded the files under the names "[myName].mid" and "[myName].wav" to https://www.dropbox.com/request/iezjutzG1kGxL75z8okS

Task 2

Arty listens to your guitar audio, transcribes the notes and playing techniques, and makes a MIDI file that will reflects them. When the virtual violin plays that MIDI file, it will follow the notes and playing techniques you used. The map between guitar and violin techniques is as below:

Right hand techniques:
- Palm-mute -> Pizzicato
- Fingerstyle -> Marcato
- Playing with pick -> Sustain

Left hand techniques:
- Guitar harmonics -> Tremolo
- Slide up from below -> Tremolo
- Vibrato -> Vibrato (for sustain and marcato)

Notes:
- Playing palm mute will always result in pizzicato, regardless of left hand technique.
- Vibrato can apply to marcato and sustain articulations (but not pizzicato)
- Tremolo is the same, regardless of playing fingerstyle or with a pick

==========
Instructions to setup the virtual violin:

Open your DAW of choice, make a MIDI track and set Plogue Sforzando as the virtual instrument. In the **decompressed** sso-small folder, drag the "Strings - Violin Solo 2 KS.sfz" file from the "Sonatina Symphony Orchestra" folder into Sforzando. It should look as below.

Plogue Sforzando with Violin patch loaded

Then, record melodies of your choice using various playing techniques into Arty. Make sure to only use single notes, not chords! Drag and drop the generated MIDI into the Sforzando track in your DAW. See if different playing techniques accurately trigger the correct articulation according to the table below. (For example, does palm mute trigger pizzicato correctly?). The current articulation is displayed in the top left of the sforzando User Interface (see picture above) *

	Never triggers	Rarely Triggers	Sometimes Triggers	Often triggers	Reliably Triggers	I didn't play that articulation
Playing with a pick, no vibrato -> Sustain (non vibrato)	○	○	○	○	○	○
Playing with a pick, vibrato -> Sustain (with vibrato)	○	○	○	○	○	○
Fingerstyle, no vibrato -> Marcato (non vibrato)	○	○	○	○	○	○
Fingerstyle, vibrato -> Marcato (with vibrato)	○	○	○	○	○	○
Palm Mute -> Pizzicato	○	○	○	○	○	○
Guitar harmonics -> Tremolo	○	○	○	○	○	○
Slide -> Tremolo	○	○	○	○	○	○

Feedback

Did you encounter the following issue with the transcribed MIDI file

- [ ] Notes I played are not in the generated MIDI (missing notes)
- [ ] The MIDI contains notes I didn't play (extraneous notes)
- [ ] The notes are in the wrong octave
- [ ] The pitch of the note is wrong (beyond octave errors)
- [ ] The timing of the notes is wrong
- [ ] Other:

How would you rate the pitch tracking *

	1	2	3	4	5	
Very inaccurate	○	○	○	○	○	Very accurate

Other comments about the pitch tracking:

Your answer

DAW/music production experience *

	1	2	3	4	5	
I've never used a DAW before	○	○	○	○	○	I produce music in DAWs regularly

Virtual instrument experience. *

- ○ I had never used virtual instruments (VSTi) before (in which case, please email me at sfranjou@mit.edu and I'll help you setup)
- ○ I have used VSTi but I never use keyswitches or MIDI CC automation
- ○ I have used keyswitches / MIDI CC / programmed articulation changes midi to make VSTi more expressive
- ○ I often program articulation switches and MIDI CC for VSTi

Would you use Arty in your regular music making ?

- ☐ I would use Arty as is
- ☐ I would use Arty if it were more accurate
- ☐ I would use Arty if it were integrated into my DAW
- ☐ I would not use Arty
- ☐ Other:

If you would use Arty, what did you like about it? If you wouldn't use Arty, why not?

Your answer

Are there playing techniques you would have liked Arty to detect but it doesn't?

Your answer

Next Clear form

Never submit passwords through Google Forms.

Bibliography

[1] C. Kehling, J. Abeßer, C. Dittmar, and G. Schuller, "Automatic tablature transcription of electric guitar recordings by estimation of score- and instrument-related parameters," in *DAFx 2014 - Proceedings of the 17th International Conference on Digital Audio Effects*, 2014. [Online]. Available: http://bandfuse.com/

[2] G. Chew, "Articulation and phrasing," 2001. [Online]. Available: https://www.oxfordmusiconline.com/grovemusic/view/10.1093/gmo/9781561592630.001.0001/omo-9781561592630-e-0000040952

[3] S. Huang, Q. Li, C. Anil, X. Bao, S. Oore, and R. B. Grosse, "Timbretron: A wavenet(cyclegan(cqt(audio))) pipeline for musical timbre transfer," *7th International Conference on Learning Representations, ICLR 2019*, 11 2018. [Online]. Available: https://arxiv.org/abs/1811.09620v2

[4] "Roland gr-500," Jun 2021. [Online]. Available: https://www.vintagesynth.com/roland/gr500.php

[5] B. Stoner, "The history of guitar synths," Jul 2021. [Online]. Available: https://www.guitarworld.com/features/history-of-guitar-synths

[6] C. Peiper, D. Warden, and G. Garnett, "An Interface for Real-time Classification of Articulations Produced by Violin Bowing," in *Proceedings of the International Conference on New Interfaces for Musical Expression NIME*, 2003, pp. 192–196. [Online]. Available: http://dl.acm.org/citation.cfm?id=1085760

[7] Z. Wang, J. Li, X. Chen, Z. Li, S. Zhang, B. Han, and D. Yang, "Musical instrument playing technique detection based on fcn: Using chinese bowed-stringed instrument as an example," 10 2019. [Online]. Available: http://arxiv.org/abs/1910.09021

[8] L. Su, L. F. Yu, and Y. H. Yang, "Sparse cepstral and phase codes for guitar playing technique classification," in *Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR 2014*, 2014, pp. 9–14. [Online]. Available: http://mac.citi.sinica.edu.tw/

[9] T. H. Özaslan, E. Guaus, E. Palacios, J. L. Arcos, T. Ozaslan, E. Guaus, E. Palacios, and J. L. Arcos, "Attack based articulation analysis of nylon

string guitar," in *Proc. of CMMR*, 2010, pp. 285–298. [Online]. Available: http://www2.iiia.csic.es/$\sim$arcos/papers/3842.pdf

[10] Y. P. Chen, L. Su, and Y. H. Yang, "Electric guitar playing technique detection in real-world recordings based on F0 sequence pattern recognition," in *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015*, 2015, pp. 708–714. [Online]. Available: http://play.riffstation.com/

[11] J. Abeßer, H. Lukashevich, and G. Schuller, "Feature-based extraction of plucking and expression styles of the electric bass guitar," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 2290–2293, 2010.

[12] T.-W. Su, Y.-P. Chen, L. Su, and Y.-H. Yang, "Tent: Technique-embedded note tracking for real-world guitar solo recordings," *Transactions of the International Society for Music Information Retrieval*, vol. 2, pp. 15–28, 7 2019. [Online]. Available: http://transactions.ismir.net/articles/10.5334/tismir.23/http://transactions.ismir.net/article/10.5334/tismir.23/

[13] E. J. Humphrey, J. P. Bello, and Y. Lecun, "Feature learning and deep architectures: new directions for music informatics," *J Intell Inf Syst*, vol. 41, pp. 461–481, 2013. [Online]. Available: http://www.music-ir.org/mirex/.

[14] H. Chen and Y. Chen, "Mitt: Musical instrument timbre transfer based on the multichannel attention-guided mechanism," pp. 568–581, 8 2021. [Online]. Available: https://link.springer.com/10.1007/978-3-030-84522-3_47

[15] A. Bitton, P. Esling, and T. Harada, "Vector-Quantized Timbre Representation," jul 2020. [Online]. Available: https://arxiv.org/abs/2007.06349v1http://arxiv.org/abs/2007.06349

[16] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable Digital Signal Processing," sep 2020. [Online]. Available: https://goo.gl/magenta/ddsp-exampleshttp://arxiv.org/abs/2001.04643

[17] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra, "Essentia: An audio analysis library for music information retrieval." International Society for Music Information Retrieval, 2013, pp. 493–498.

[18] A. de Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, pp. 1917–1930, 4 2002. [Online]. Available: http://asa.scitation.org/doi/10.1121/1.1458024

[19] R. McNab, L. A. Smith, and I. H. Witten, "Signal processing for melody transcription," 1 1996, pp. 301–307.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] J. Salamon and E. Gomez, "Melody extraction from polyphonic music signals using pitch contour characteristics," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, pp. 1759–1770, 2012.