BUILDING THE METAMIDI DATASET: LINKING SYMBOLIC AND AUDIO MUSICAL DATA

Jeff Ens
Simon Fraser University
jeffe@sfu.ca

Philippe Pasquier
Simon Fraser University
pasquier@sfu.ca

ABSTRACT

We introduce the MetaMIDI Dataset (MMD), a large scale collection of 436,631 MIDI files and metadata. MMD contains artist and title metadata for 221,504 MIDI files, and genre metadata for 143,868 MIDI files, collected during the web-scraping process. MIDI files in MMD were matched against a collection of 32,000,000 30-second audio clips retrieved from Spotify, resulting in over 10,796,557 audio-MIDI matches. In addition, we linked 600,142 Spotify tracks with 1,094,901 MusicBrainz recordings to produce a set of 168,032 MIDI files that are matched to the MusicBrainz database. We also provide a set of 53,496 MIDI files using audio-MIDI matches where the derived metadata on Spotify is a fuzzy match to the web-scraped metadata. These links augment many files in the dataset with the extensive metadata available via the Spotify API and the MusicBrainz database. We anticipate that this collection of data will be of great use to MIR researchers addressing a variety of research topics.

1. INTRODUCTION

Large-scale metadata-rich MIDI datasets containing audio-MIDI matches [1–3] are indispensable in a wide variety of research contexts. For example, the Lakh Midi Dataset (LMD) [3] has been applied in many different contexts, including training generative music systems [4, 5], tempo-estimation [6], genre classification [7] and even as a primary data-source for new datasets [8, 9]. Motivated by the widespread demand for datasets of this nature, we created the MetaMIDI Dataset (MMD), which contains 2.4 times the number of MIDI files in the LMD, and audio-MIDI matches associating MIDI files with Spotify and MusicBrainz. To put the following numbers into context, we note that there is a many-to-one relationship between Spotify track ids and the actual audio recording. In this paper, we describe the process of assembling the dataset, which consists of the following contributions:

- Collection of 436,631 MIDI files.
- Scraped artist + title metadata for 221,504 MIDIs (10 times more than the LMD).
- Scraped genre metadata for 143,868 MIDIs.
- An improved audio-MIDI matching procedure, which produced 10,796,557 audio-MIDI matches linking 237,236 MIDIs to one or more tracks on Spotify.
- 829,728 high reliability audio-MIDI + scraped metadata (artist and title) matches linking 53,496 MIDIs to one or more tracks on Spotify.
- A method for linking Spotify tracks and MusicBrainz recordings, producing 8,263,482 unique links that associate 1,094,901 MusicBrainz recordings with 600,142 Spotify tracks.
- 168,032 MIDIs matched to MusicBrainz IDs via the Spotify/MusicBrainz linking procedure.

2. DATA COLLECTION

We scraped publicly available websites and were able to amass a collection of 436,631 unique MIDI files. Candidate websites were selected using a search engine to query various phrases including keywords such as MIDI, music, and a variety of musical genres. A list of the sites scraped and the number of MIDI files found on each site is provided in the dataset. Where possible, we also collected additional metadata, such as the artist, title and genre of associated with a particular MIDI file.

3. AUDIO MIDI MATCHING

To augment MMD with additional metadata, we match the MIDI files against a large metadata-rich collection of audio clips. Although the LMD is comprised of audio-MIDI matches against the Million Song Dataset [10], we decided to use 30-second preview clips made available through the Spotify API [1]. The primary motivation for this decision was the fact that the Spotify API provides over an order of magnitude more data. Using the Spotify API, we were able to collect 32,000,000 30-second MP3 files (over 13TB of raw data). To compute the audio-MIDI matches, we model our approach after the procedure employed by Raffel [3], who matched the 176,581 MIDI files in the LMD with 1,000,000 audio files in the Million Song Dataset [10]. However, we make some modifications to the matching algorithm to accommodate the large amount of data which was collected.

[1] https://developer.spotify.com/documentation/web-api/

Raffel's audio-MIDI matching procedure is comprised of two stages [3]. In the first stage, which we refer to as the blocking stage, audio-MIDI pairs which are unlikely to be a match are removed from consideration. In the second stage, which we refer to as the matching stage, a confidence score (on the range [0,1]) is computed for each remaining audio-MIDI pair. To be considered a valid match, the audio-MIDI pair must have a confidence score greater than 0.5. To compute the confidence score for an audio-MIDI pair, Raffel computes the Constant-Q Transform (CQT) [11] for the audio file and the audio-rendered MIDI file, using 48 logarithmically-spaced bins from C2 to B5 (12 bins per octave). Then, the dynamic time warping (DTW) algorithm is used to find the optimal alignment, from which the confidence score is directly computed [3]. Although this procedure produces good results, it is extremely slow, as DTW has quadratic run-time, which makes this approach intractable.

To speed up the matching process, Raffel proposes learning distance preserving low-dimension embedding spaces, which should allow for highly dissimilar matches to be efficiently removed from the search space. Raffel explores two approaches, an attention-based network ($\mathcal{H}_\infty$) that embeds arbitrary length CQT matrices into a 128-bit hash code [12], and a convolution-based network ($\mathcal{H}_k$) that maps $k \times 48$ CQT matrices into 32-bit hash codes [13], which can be used to transform a $n \times 48$ CQT matrix into a sequence of $\lfloor \frac{n}{k} \rfloor$ 32-bit hash codes. Using trained embedding networks $\mathcal{H}_\infty$ and $\mathcal{H}_8$, Raffel employs the following procedure to match a single MIDI CQT m against a set of audio CQTs A.

1. Blocking Stage
   (a) Compute $\mathcal{D}_{\mathtt{H}}(\mathcal{H}_\infty(a), \mathcal{H}_\infty(m))$ for each $a \in A$, where $\mathcal{D}_{\mathtt{H}}$ is the bitwise hamming distance.
   (b) Construct a set A', containing the $t_1 = 100{,}000$ $a \in A$ that are closest to m, using the distances calculated in 1a.
   (c) Compute $\mathtt{DTW}(\mathcal{H}_8(a), \mathcal{H}_8(m))$ for each $a \in A'$.
   (d) Construct a set A'' containing the $t_2 = 250$ $a \in A'$ that are closest to m, using the distances calculated in 1c.
2. Matching Stage
   (a) Compute $\mathtt{DTW}(a, m)$ for each $a \in A''$ and record any matches with more than .5 confidence.

3.1 Modifications to the Matching Procedure

According to Raffel's measurements, it takes an average of 108 seconds on a single CPU to match one MIDI file against 1,000,000 Audio files. As a result, without making modifications to Raffel's procedure, it would take roughly 558 days on a 32-core CPU to match our collections of audio and MIDI files. In order to optimize the audio-MIDI matching procedure to our specific context, we make changes to the blocking stage. Notably, since we do not modify the second stage, and use Raffel's code [2] to compute the confidence scores, our matches can be considered to be the same quality as those found in the LMD.

The simplest modification involved implementing a c++ version of the DTW code for 32-bit hash sequences, used in the blocking stage, which runs 2 times faster than Raffel's jit-compiled Cython implementation according to our measurements. We also reconsider the use of the attention based embedding network $\mathcal{H}_\infty$ in Steps 1a and 1b. Using Raffel's approach, Step 1a can be computed very quickly, accounting for less than 1% of the total algorithm run-time. However, due to the low reliability of distance measurements in this embedding space, relatively few audio files can be removed from consideration. As a result, Step 1c takes much longer to run, accounting for roughly half of the total run-time. One reason for the limited accuracy of this approach, is that $\mathcal{H}_\infty$ must embed MIDI and audio CQTs into the same 128-bit hash code, despite MIDI files being much longer than the audio files.

To address this issue, we use $\mathcal{D}_{\mathtt{W}}$, defined in Eq. 1, to compute the distances in Step 1a. Given an $n \times 48$ MIDI CQT m and an audio CQT a, we build a set of 30-second length sub-sequences ($\mathcal{X}^m$) from m, as defined in Eq. 1a, where s is the stride. Using $\mathcal{H}_{128}$ we map each 30-second length CQT matrix (i.e. 646×48) x to a hash code by splitting x into contiguous windowed sub-sequences, computing $\mathcal{H}_{128}(\cdot)$ for each sub-sequence, and concatenating the resulting hash codes. Formally, we refer to this process as $\mathcal{H}^\star_k$, which we define in Eq. 1b, where $\oplus$ denotes concatenation. Then, as shown in Eq. 1c, we compute the bitwise hamming distance ($\mathcal{D}_{\mathtt{H}}$) between $\mathcal{H}^\star_{128}(x)$ and $\mathcal{H}^\star_{128}(a)$ for each $x \in \mathcal{X}^m$, considering the minimum distance to be representative of the distance between m and a.

$$\mathcal{X}^m = \{m[si{:}si{+}646] \,:\, 0 \leq i < \left\lfloor \frac{n-646+1}{s} \right\rfloor\} \quad \text{(1a)}$$

$$\mathcal{H}^\star_k(x) = \oplus\{\mathcal{H}_k(x[ki{:}k(i{+}1)]) : 0 \leq i < \left\lfloor \frac{||x||}{k} \right\rfloor\} \quad \text{(1b)}$$

$$\mathcal{D}_{\mathtt{W}}(a,m) = \min(\{\mathcal{D}_{\mathtt{H}}(\mathcal{H}^\star_{128}(a), \mathcal{H}^\star_{128}(x)) : x \in \mathcal{X}^m\}) \quad \text{(1c)}$$

3.2 Training the Embedding Networks

We derive our neural network architecture from the one used by Raffel [3]. The first section of the network is comprised of k groups, with each group is containing 2 3×3 convolutional layers, followed by a 2×1 max pooling layer. The second section contains two dense layers with 2048 units each, followed by a 32-dimensional output. The ReLU activation is used in all layers, except for the last layer, which uses the tanh activation function to effectively binarize the output. For the $\mathcal{H}_{128}$ network, which learns to downsample a sequence of 128×48 CQT matrix into a 32-bit hash code, there are $k = 5$ groups, using the filter sizes 64, 64, 64, 32, and 16 for each group respectively. For the $\mathcal{H}_8$ network, which learns to downsample a 8×48 CQT matrix into a 32-bit hash code, there are $k = 3$ groups, using 64, 32, and 16 filters per group respectively. We train $\mathcal{H}_{128}$ and $\mathcal{H}_8$ using the same triplet loss as Raffel.

[2] https://github.com/craffel/midi-dataset

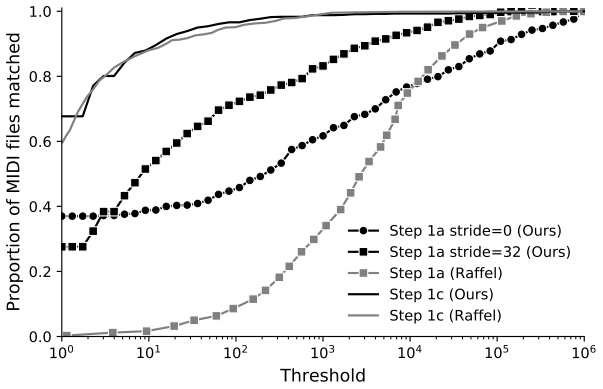

Figure 1. Percentage of MIDI files matched at thresholds.

In terms of training data, we use the 116,189 audio-MIDI matches from the LMD, which we split into testing, validation and training datasets. We train each network with a learning rate of $1\text{e}{-4}$, and early stopping on validation every 1000 batches, using Keras [14].

3.3 Evaluating the Embedding Networks

To evaluate the expected accuracy of distance calculations using our trained embedding networks, we use the same method proposed by Raffel. For a known audio-MIDI pair (m, a), we measure the distance between m and a set of 1,000,000 audio files $\mathcal{X}$, with $a \in \mathcal{X}$, to determine the rank of the correct match. After repeating this process for 1,000 audio-MIDI pairs in our test set, we can measure the proportion of MIDI files where the correct match ranks below a particular threshold. The results are presented in Figure 1, including results previously presented by Raffel for purposes of comparison [3]. Although Raffel used different data to train and evaluate the embedding, we can be fairly confident in the reliability of our comparison, as the curve for our $\mathcal{H}_8$ embedding network (Step 1c (Ours)) is nearly identical to the curve for Raffel's $\mathcal{H}_8$ embedding network (Step 1c (Raffel)). Although using $\mathcal{D}_{\mathbb{W}}$ slows down Step 1, the results demonstrate that it is much more accurate, which means we can reduce the number of comparisons needed in Steps 1c and 1d, which ultimately speeds up the algorithm, as Step 1c accounts for roughly half of the total run-time.

3.4 Matching Against 32,000,000 Audio Files

Clearly, a large factor contributing to the run-time of the matching algorithm is the threshold levels (t_1, t_2) for each stage of the search. Raffel et al. determine t_1 and t_2 based on the evaluation method presented in Figure 1. However, this approach is merely a proxy for what we are actually trying accomplish. Put simply, in matching a large collection of MIDI files with a large collection of Audio Files, we are trying to maximize the number of matches. In order to get a sense of the relationship between run-time and the number of matches, we run our matching procedure with 1,000 MIDI files and 10,000,000 audio files, using various thresholds. The results in figure 2 show that we pay a high computational cost to increase the number of MIDIs matched. For example, increasing the thresholds from $t_1 = 100{,}000$ / $t_2 = 250$ to $t_1 = 1{,}000{,}000$ / $t_2 = 2{,}500$ increases the run-time by 560%, while only yielding a 10% increase in the number of MIDIs matched and a 200% increase in the number of Audio files matched.

Due to memory limitations, it is not possible to match a MIDI CQT against all 32,000,000 audio CQTs at once. As a result, we subdivide the audio CQTs into four chunks, and process them each separately. In light of the results in the previous section, we decided to set $t_1 = 100{,}000$ and $t_2 = 250$ for each chunk. In Table 1, we report the results of the Audio-MIDI matching procedure. In comparison to the LMD, where only 26% of the MIDI files were matched to at least one Audio file, we were able to match 56% of the MIDI files, for a total of 237,236 MIDI files matched. Notably, our modifications to the matching procedure also had a substantial impact on the run-time, as the average run-time per match was only 3.3 times more than the run-time for LMD matching, despite matching against over 32 times more audio.

3.5 High Reliability Audio-MIDI Matches

Although the audio-MIDI matches are fairly reliable, Raffel notes that it is not uncommon for there to be false positives when an audio-MIDI pair share the same chord progression [3]. To address these issues, we produce a subset of the audio-MIDI matches which are more reliable, using artist+title metadata that was collected during the scraping process. In short, we only retain audio-MIDI matches where the title or artist scraped with the MIDI file is a fuzzy match to the metadata on Spotify. Since artists and title metadata frequently contain extraneous information, we remove all content in parenthesis or square brackets, and remove all content following a dash. As a result, the Spotify track titled "Rain Is Falling (Karaoke Version) - Originally Performed By Electric Light Orchestra" would be reduced to "Rain is Falling" after pre-processing. We measure the similarity between two strings using cosine similarity on their tri-gram profiles, and only keep matches when the similarity exceeds .8 for either the artist or the title metadata. Once this procedure has been completed, we are left with 53,496 (12%) matched MIDI files and 829,728 total matches.

4. LINKING SPOTIFY AND MUSICBRAINZ

To further expand the dataset, we make links between Spotify track ids to MusicBrainz recording ids using a classifier trained on audio features. Although AcousticBrainz Labs has provided an archive [3] of the Echo Nest mappings between MusicBrainz and Spotify, we were only able to match 24,363 MIDI files to MusicBrainz IDs using this resource. To train our classifier, we gathered a set of ground truth data using International Standard Recording Codes (ISRC), which are provided by both Spotify and MusicBrainz. Although Spotify provides this information for almost all of their tracks via their API, only a percentage of recordings in the MusicBrainz database have been

[3] https://labs.acousticbrainz.org/million-song-dataset-echonest-archive/

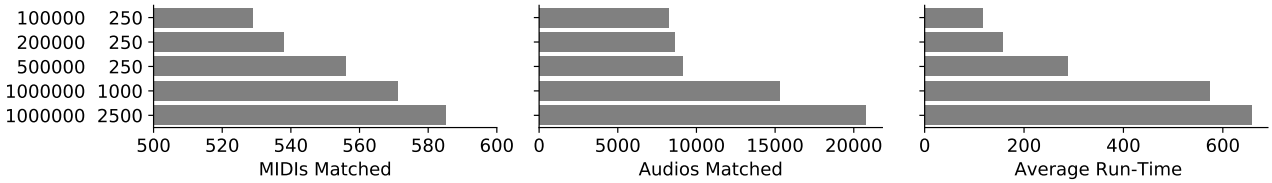

Figure 2. The number of MIDI files matched, Audio recordings matched and average match run-time for different thresholds. On the left, the first value denotes t_1 and the second value denotes t_2.

Dataset	MIDIs	Audio Source	Matching Method	Matched MIDIs	Matched Audios	Total Matches	Percentage of MIDIs Matched
LMD	176,581	MSD [10]	Audio	45,129	31,034	116,189	25.6%
MMD	436,631	Spotify	Audio	237,236	2,209,941	10,796,557	52.7%
MMD	436,631	Spotify	Audio + Text	53,496	347,703	829,728	12.3%
MMD	436,631	MusicBrainz⋆	Audio	168,032	1,094,901	8,384,256	38.5%
MMD	436,631	MusicBrainz⋆	Audio + Text	34,174	408,922	1,232,909	7.8%

Table 1. Statistics for the audio-MIDI matching. Note that the MusicBrainz matches were computed by combining the Spotify audio-MIDI matches and the Spotify-MusicBrainz links (Section 4). The Percentage of MIDIs Matched column reports the percentage of MIDI files in the respective dataset that have at least one match to an audio file. Total Matches denotes the total number of unique audio-MIDI pairs matched.

labeled with an ISRC code. Using the ISRC codes which were available, we were able to compile about 100,000 unique ground truth matches. This data was divided into training, validation and testing sets.

We use the AcousticBrainz API [4] to obtain features for recordings in the MusicBrainz database, since the actual audio is not provided by MusicBrainz or AcousticBrainz. To extract features from the 30-second Spotify preview clips, we use the same feature extractor as AcousticBrainz (Essentia [15]). Using the low-level features extrated via Essentia, we obtain a feature vector of dimension 1773 to represent each audio clip. Then we trained a classifier to predict whether a pair of vectors, one collected from the AcousticBrainz database, and another from Spotify, correspond to the same recording. To train the classifier, we expose the model to ground truth matches, where the AcousticBrainz recording and Spotify recording share the same ISRC, and negative matches, where both recordings do not share the same ISRC. To construct a negative match, we randomly select one recording from each data source (AcousticBrainz and Spotify). Note that for training, validation and testing we make sure the model is exposed to both conditions (ground truth and negative match) an equal number of times.

We use the XGBoost library [16] to train a gradient boosting model. To determine the optimal hyperparameters for the model, we perform a grid search using the following parameters: nestimators $\{2500, 5000\}$, learning rate $\{.1, .25, .5, .75\}$, and max depth of $\{2, 3, 4\}$. To evaluate the models, we calculate the accuracy with which the model was able to predict if the pair of recordings was a positive (ground-truth) or negative match. We found the model with nestimators=2500 learning rate=.25 and max depth=4 to perform the best on the validation set, achieving 97.6% accuracy. To give us some indication that we are not simply over-fitting on the validation set, we compute the accuracy of the best model using the testing set. Based on the fact that the best model scored 97.5% accuracy on the test set, which was only used once, we can be fairly confident that the model will generalize with this level of accuracy.

Since, at the time of writing, there are 5,534,103 unique recordings in the AcousticBrainz dataset, and 2,209,941 Spotify audio previews (see Table 1) which we want to match against, collecting the model's predictions for each pairwise match would be extremely computationally expensive. To make this process feasible, we first match all the artists in the MusicBrainz database against a list of artists from Spotify using tri-gram cosine distance with a threshold of .7. Then we match each the titles of each recording if the artists were a match, once again using tri-gram cosine distance with a threshold of .7. Then for each potential match, we use the classifier to predict whether it is actually an audio match. Consequently, the error rate should be lower than 2.5% since matches must also have similar metadata (artist title) to be considered a match. The entire process took about 3 days on a single computer. In Table 2 below, we outline the results of the Spotify-MusicBrainz linking process. We provide details on the MIDI-MusicBrainz matches which were derived from the audio-MIDI matches in Table 1.

5. ANALYZING THE DATASET

5.1 Overview Statistics for the Midi Files

In order get a sense of the type of data that was collected, we compute the distributions for several features. We parse a MIDI file into a set of tracks, where a track is simply the set of note onsets and offsets belonging to a $(\texttt{MIDI track}, \texttt{channel}, \texttt{instrument})$ tuple. Each track is subdivided into a sequence of bars, using the time

[4] https://acousticbrainz.org/data

	Matched Spotify IDs	Matched MusicBrainz IDs	Spotify-MusicBrainz Matches	MIDI-MusicBrainz Matches
MSD Echo Nest Archive	1,307,152	675,240	3,168,164	24,363
ISRC Matches	104,404	69,006	104,404	82,951
Ours	600,142	1,094,901	8,263,482	168,032

Table 2. Statistics for the Spotify-MusicBrainz matching.

signature information present in the MIDI file. Due to space limitations, we present a few of the most pertinent features below, providing a more comprehensive overview elsewhere [5] :

1. **Number of Tracks** : The number of tracks, as defined above, in a MIDI file.
2. **Beat Length** : The total length in quarter note beats of an entire MIDI file.
3. **Notes Per Bar** : The number of note onsets occurring in a bar. We measure this on each track separately, so that we do not conflate notes per bar and number of tracks.

We compute the distribution of each of these features across three different sets of data: the LMD, MMD, and their symmetric difference MMD Δ LMD. These distributions are shown in Figure 3. On a whole, the graphs demonstrate that LMD, MMD Δ LMD and MMD are all fairly similar, however there are some differences worth noting. The two most obvious differences, are the beat length and number of tracks. The difference in beat length distributions can mainly be explained by the fact that two of the sites we scraped MIDIs from only provide 30s preview MIDI clips for free. Since the musical quality of these shorter MIDIs is comparable to that found in the LMD, we saw no real reason to exclude these files. The difference in track counts per MIDI does not have an obvious explanation, but is worth noting nonetheless.

5.2 Estimating the Reliability of Scraped Metadata

To gauge the reliability of the scraped metadata, we analyze instances where metadata was collected for the same MIDI file (md5 checksum) from multiple sources. In total, there are over 10,000 MIDI files which satisfy this criteria. For each of these MIDI files, we compare the title/artist and genre/category metadata separately. For the title/artist metadata, we concatenate this metadata into a single string, delimited by a "-", and compute cosine similarity on their tri-gram profiles. For the genre metadata, we compute tri-gram cosine similarity between each pairwise combination of elements between two genre/category lists, and report the maximum similarity. The mean similarity is 73.7% for title/artist metadata and 1.1% for genre metadata. Immediately apparent, is the significant discrepancy, as title/artist metadata appears to be fairly consistent from site to site, while genre metadata is not. Further manual analysis reveals that the genres/categories are often very generic, which may make them unsuitable for some purposes. In some respects, this is not altogether surprising, as determining the genre/category of a piece of music is a highly subjective process, and other research has shown a significant level of disagreement [17]. However, with regards to the artist/title metadata, these results seem to indicate that we can be fairly confident in this form of metadata. It is worth noting that this type of analysis does not rule out cases where artist/title metadata on multiple sites was derived from a single inaccurate source to begin with.

5.3 False Positives and Audio Midi Matching

Using the standard and high-reliability sets of audio-MIDI matches, we can further analyze the source of false positives in the matching procedure. To do this, we compare the genre distribution of each set of audio-MIDI matches. Since Spotify uses more than 5,000 genres, many of which contain descriptors of particular locations (ex. Louisville Indie) or languages (ex. Spanish Indie Pop), we preprocess the data to remove geographical locations, demonyms and languoids. This results in about 2,500 genres. To further aggregate these genres into broader categories we employ a graph embedding approach. Using the Spotify API, we collect a list of genres for 336,507 different artists. For example, the band U2 has a genre list containing three genres: Irish Rock, Permanent Wave, and Rock. Note that after we apply our pre-processing procedure, U2 has two genres: Rock and Permanent Wave. Of particular interest for our purposes here, is artists which have a genre list containing more than one genre, as the overall frequency with which two genres co-occur within genre lists should provide a good indication of their similarity.

Then we construct a graph where each genre is a node, and the edge weights between nodes are the count of co-occurrences within the genre lists. To create the embedding, we use the Node2Vec algorithm [18], which creates an embedding space that is trained on relations found within the graph. Similar to the word2vec algorithm [19], where adjacent groupings of words inform the embedding, random walks on the graph are used to infer a context for each node. We use the nodevectors [6] implementation of Node2Vec to learn a 32-dimensional embedding space, training with random walks of length 30 for 100 epochs. To determine a small set of k representative genres, we use Agglomerative Hierarchical Clustering with Ward linkage to partition the embedded genre vectors into k clusters. In order to give each cluster a human-readable label, we count the frequency with which each of the genres belonging to the cluster is used in the genre lists. The most frequently used genre is taken as the label for each genre. We set $k = 15$, which produces the following set of genres: indie, rock, experimental, jazz, pop, metal, musica, electronic,

[5] https://github.com/jeffreyjohnens/MetaMIDIDataset

[6] https://github.com/VHRanger/nodevectors

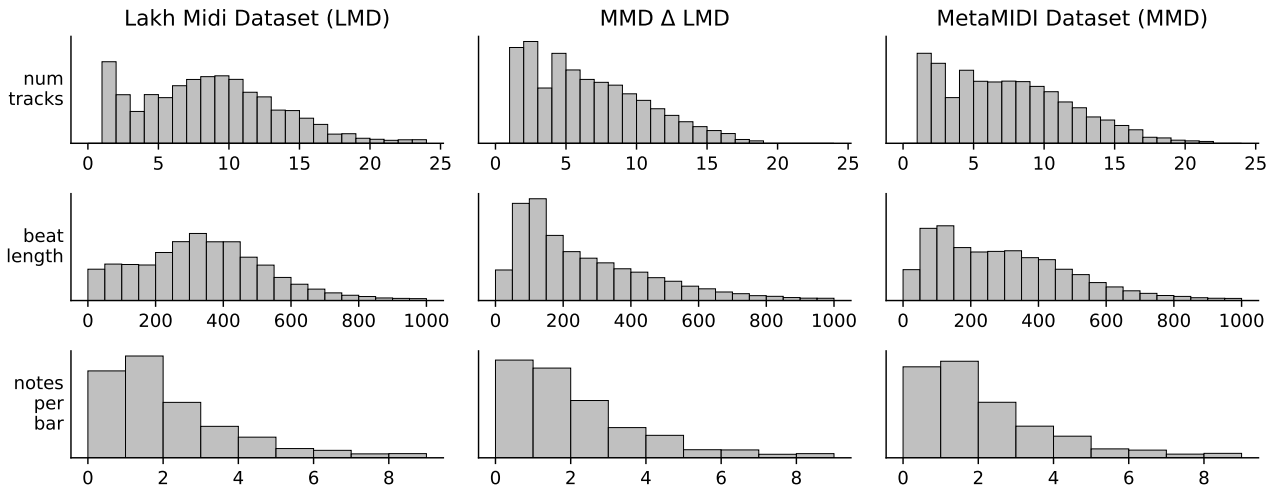

Figure 3. The distributions for various features computed on LMD, MMD Δ LMD and MMD.

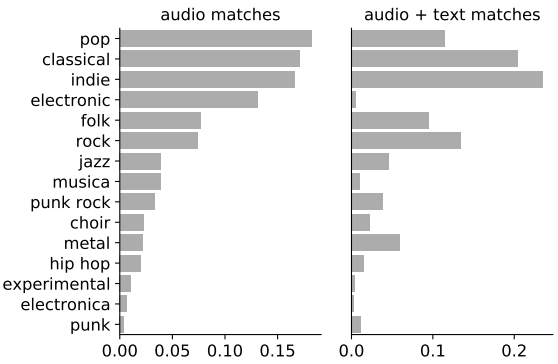

Figure 4. The distribution of genres for matched MIDI files using two methods: audio and audio + text.

folk, choir, classical, punk, punk rock, hip-hop, and electronica. We admit that our decision to set $k = 15$ is fairly arbitrary, however, due to the nature of our clustering procedure, selecting a different value for k would not have a large impact. For example, setting $k = 16$ produces the same set of 15 genres with one new genre cluster.

In Figure 4 the genre distributions are plotted for each version of the matching procedure. Since we can be fairly confident that the audio + text matches are more accurate, analyzing the discrepancies between the genre distributions can help identify some of the shortcomings of the `DTW` audio-MIDI match algorithm. In the audio matches distribution, we see a large increase in pieces classified as pop and electronic, which indicates these pieces are likely the source of most of the error. This may be a byproduct of their simple harmonic structure, and/or the prevalence of remixes and covers within these particular genres.

6. USING THE METAMIDI DATASET

The dataset, as well as a detailed description of its contents, can be accessed through the MetaMIDI Dataset repository [7]. Throughout the dataset, MIDI files are identified by their md5 checksum. We provide mappings from md5 checksums to Spotify track ids and MusicBrainz recording ids, which can be used to access a plethora of metadata. The MusicBrainz database provides access to variety of linked entities including artists, recordings, releases, composers, producers, recording engineers and labels. Detailed attributes are available for most entities. For example, the MusicBrainz entry for the group Bon Iver, provides the date and location where the group was established, a list of aliases, a set of genre tags, and a comprehensive list of links to external websites. Using the Spotify API, a variety of metadata can be accessed, including track-based audio features such as danceability, valence, liveness and energy; and additional metadata ranging from genre to artist popularity.

7. CONCLUSION

Although the primary contribution is the dataset itself, we have also provided reusable insights related to the audio-MIDI matching algorithm and the Spotify-MusicBrainz linking procedure. One limitation worth noting, is the uncertainty in relying on Spotify's 30-second clips to persist into the future, which unfortunately has already become an issue with the 7Digital clips in the Million Song Dataset [10]. With regards to the dataset, we anticipate a wide variety of potential use-cases for this data. Since many generative systems have been trained using the Lakh MIDI Dataset [4, 5, 9], the MMD will undoubtedly be a valuable asset to research in this area, as it features 2.4 times more MIDI files. More broadly, the metadata that our audio-MIDI matches provide access to, as well as the audio-MIDI matches themselves, can be used to support a variety of scientific inquires related to MIR and Musicology.

[7] https://github.com/jeffreyjohnens/MetaMIDIDataset

8. ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Helmut & Hugo Eppich Family Graduate Scholarship.

9. REFERENCES

[1] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the MAESTRO dataset," *arXiv preprint arXiv:1810.12247*, 2018.

[2] F. Foscarin, A. Mcleod, P. Rigaux, F. Jacquemard, and M. Sakai, "ASAP: a dataset of aligned scores and performances for piano transcription," in *Proc. of the 21st International Society for Music Information Retrieval Conference*, 2020.

[3] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching," Ph.D. dissertation, Columbia University, 2016.

[4] C. Donahue, H. H. Mao, Y. E. Li, G. W. Cottrell, and J. McAuley, "LakhNES: Improving multi-instrumental music generation with cross-domain pre-training," in *Proc. of the 20th International Society for Music Information Retrieval Conference*, 2019, pp. 685–692.

[5] A. Roberts, J. H. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," in *Proc. of the 35th International Conference on Machine Learning*, 2018, pp. 4361–4370.

[6] H. Schreiber and M. Müller, "A single-step approach to musical tempo estimation using a convolutional neural network." in *Proc. of the 19th International Society for Music Information Retrieval Conference*, 2018, pp. 98–105.

[7] A. Ferraro and K. Lemström, "On large-scale genre classification in symbolically encoded music by automatic identification of repeating patterns," in *Proc. of the 5th International Conference on Digital Libraries for Musicology*, 2018, pp. 34–37.

[8] E. Manilow, G. Wichern, P. Seetharaman, and J. Le Roux, "Cutting music source separation some slakh: A dataset to study the impact of training data quality and quantity," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2019, pp. 45–49.

[9] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proc. of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 34–41.

[10] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proc. of the 12th International Society for Music Information Retrieval Conference*, 2011.

[11] J. C. Brown, "Calculation of a constant q spectral transform," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.

[12] C. Raffel and D. P. Ellis, "Pruning subsequence search with attention-based embedding," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2016, pp. 554–558.

[13] ——, "Large-scale content-based matching of midi and audio files." in *Proc. of the 16th International Society for Music Information Retrieval Conference*, 2015, pp. 234–240.

[14] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[15] D. Bogdanov, N. Wack, E. Gómez Gutiérrez, S. Gulati, H. Boyer, O. Mayor, G. Roma Trepat, J. Salamon, J. R. Zapata González, X. Serra *et al.*, "Essentia: An audio analysis library for music information retrieval," in *Proc. of the 14th International Society for Music Information Retrieval Conference*, 2013.

[16] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[17] R. Brisson and R. Bianchi, "On the relevance of music genre-based analysis in research on musical tastes," *Psychology of Music*, vol. 48, no. 6, pp. 777–794, 2020.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.