Melody Identification in Standard MIDI Files

Zheng Jiang
Carnegie Mellon University
zjiang1@andrew.cmu.edu

Roger B. Dannenberg
Carnegie Mellon University
rbd@cs.cmu.edu

ABSTRACT

Melody identification is an important early step in music analysis. This paper presents a tool to identify the melody in each measure of a Standard MIDI File. We also share an open dataset of manually labeled music for researchers. We use a Bayesian maximum-likelihood approach and dynamic programming as the basis of our work. We have trained parameters on data sampled from the million song dataset [1, 2] and tested on a dataset including 1703 measures of music from different genres. Our algorithm achieves an overall accuracy of 89% in the test dataset. We compare our results to previous work.

1. INTRODUCTION

When we listen to a piece of music, the melody is usually the first thing that catches our attention. Therefore, the identification of melody is one of the most important elements of music analysis. Melody is commonly understood to be a prominent linear sequence of pitches, usually higher than harmonizing and bass pitches. The concept of melody resists formalization, making melody identification an interesting music analysis task. Melody is used to identify songs. Often, other elements such as harmony and rhythm are best understood in relation to melody.

Many music applications depend on melody, including Query-by-Humming systems, music cover song identification, emotion detection [3], and expressive performance rendering. Many efforts in automatic composition could benefit from training data consisting of isolated melodies.

There has been a lot of research on extracting melody from audio [4]. The problem is generally easier for MIDI than audio because at least notes are already identified and separated. However, compared to audio, there seems to be less research on melody extraction. Most of the research on MIDI melody is on channel-level identification. This paper will propose an algorithm combining Bayesian probability models and dynamic programming to extract melody at the measure level.

2. RELATED WORK

In the field of symbolic files, *Skyline* is a very simple algorithm proposed by Uitdenbogerd [5]. In brief, the idea of this method is to pick the highest pitch at any moment as belonging to the melody. Chai and Vercoe offer an enhanced version of this approach [6]. In pop music, we observe that there are often accompaniment notes above the melody line, leading to failure of the Skyline algorithms. Uitdenbogerd presents three more methods [4]: 1) Top Channel (choose the channel with the highest mean pitch), 2) Entropy Channel (choose the channel with the highest entropy), and 3) Entropy Part (segment first, then use Entropy Channel). Shan [7] proposed using greatest volume (MIDI velocity) because melody is typically emphasized through dynamics. Li et al. identify melodies by finding common sequences in multiple MIDI files, but this obviously requires multiple versions of songs [8]. Li, Yang, and Chen [9] use a Neural Network and features such as chord rate, pronunciation rate, average note pitch, instrument, etc., trained on 800 songs to estimate the likelihood that a channel is the melody channel. Velusamy, et al. [10] use a similar approach, but prune notes that do not satisfy certain heuristics and use a hand-crafted linear model for ranking channels [9]. Rizo, et al [11] introduces an algorithm to identify the track that contains the melody using statistical properties of the musical content and machine learning techniques.

All of these algorithms assume that the melody appears on one and only one channel, so the problem is always to select one of up to 16 channels as the melody channel. Depending on the data, this can be a frequent cause of failure, since the melody can be expressed by different instruments in different channels at different times. An interesting approach is *Tunerank* [12], which groups and labels notes according to harmony and dissonance with other notes, pitch intervals between consecutive notes, and instrumentation, without assuming the melody is in only one channel.

Previous work is hard to evaluate based on publications, with accuracy reports ranging from 60% to 97%, no labeled public datasets, and few shared implementations. The properties of music arrangements and orchestrations in MIDI files can cause many problems. The simplest case, often assumed in the literature, is that the melody appears in one and only one channel. At least four more complex conditions are often found: 1) The melody is sometimes played in unison or octaves in another channel, 2) the melody switches from one instrument (channel) to another from one phrase or repetition to another, 3) the melody is fragmented across channels even within a single measure (this happens but seems to be rare), and 4) there are multiple overlapping melodies as in counterpoint, rounds, etc.

3. DATASET

In most related work, published links to datasets have expired, so we collected and manually labeled a new dataset which contains the training data of 5823 measures in 51 songs and test data of 1703 measures in 22 songs. For each song in the training data, the melody is mostly all on one channel, which is labeled as such. (This made labeling much easier, but we had to reject files where the melody appeared significantly in multiple channels or there is no melody at all.) In the test data, the melody is not constrained to a single channel, and each measure of the song is labeled with the channel that contains the melody. Measure boundaries are based on tempo and time signature information in the MIDI file. The MIDI files are drawn from songs in the Lakh dataset [1]. This collection contains MIDI files that are matched to a subset of files in the million song dataset [2]. We used tags there to limit our selection to pop songs.

The test data is collecting from Chinese, Japanese, and American pop songs. We specifically chose popular music because melody is usually present and there is usually a single melody. In addition, we hope to use this research in learning about melody structure in popular music.

It might be noted that there are many high quality MIDI files of piano music. Since all piano notes are typically on one channel, this can make the melody identification or separation a more challenging problem, and different techniques are required. We assume that in our data, once the channel containing the melody is identified, it is fairly easy to obtain the melody. Either the melody is the only thing present in the channel, or the melody is harmonized, and the melody is obtained by removing the lower notes using the Skyline algorithm.

4. ALGORITHM

Our problem consists of labeling each measure of a song with the channel that contains the melody. (It would be useful also to allow non-labels, or *nil*, indicating there is no melody, but our study ignores this option.) The algorithm begins with a Bayesian model to estimate $\boldsymbol{M}_{m,c}$, the probability that channel c in measure m contains the melody. The estimation uses features that are assumed to be jointly normally distributed and independent. Features are calculated from the content of each channel, considering the measure itself and N previous and subsequent measures, with $N \in 0, 1, ...$ In all of our experiments, we assume that melody *never* appears on channel 10, which is used for drums in General MIDI.

Although we could stop there and report the most likely channel in each measure,

$$c_m = \underset{c}{\operatorname{argmax}}\, \boldsymbol{M}_{m,c} \tag{1}$$

this does not work well in practice. There are many cases where a measure of accompaniment, counter-melody or bass appears to be more "melody-like" than the true melody. (For example, the melody could simply be a whole note in some measures.) However, it is rare for the melody to switch from one channel to another because typically the melody is played by one instrument on one channel. Channel switches are only likely to occur when the melody is repeated or on major phrase boundaries.

We can consider the melody channel for each measure, c_m, as a sequence of hidden states and per-measure probabilities as observations. We wish to find the most likely overall sequence c_m according to the per-measure probabilities, and taking into account a penalty for switching channels from one measure to the next. We model the probability of the hidden state sequence c_m as:

$$P(c_m) = \prod_m \boldsymbol{M}_{m,c_m} S_{c_{m-1},c_m} \tag{2}$$

where $S_{c_{m-1},c_m} = 1$ if there is no change in the channel ($c_{m-1} = c_m$), and S_{c_{m-1},c_m} is some penalty less than one if there is a channel change ($c_{m-1} \neq c_m$). Thus, channel switches are allowed from any measure to the next, but channel switches are considered unlikely, and any labeling that switches channels frequently is considered highly unlikely.

The parameters of this model must be learned, including: statistics for features used to estimate $\boldsymbol{M}_{m,c}$, the best feature set, the number of neighboring measures N to use in computing features, and the penalty S for changing channels. We select the feature set and compute feature statistics using our training dataset, and we evaluate their performance and sensitivity to N and S using the test dataset.

4.1 Bayesian Probability Model

The probability of melody given a set of features is represented by Equation 3, where C_0 is the condition that the melody is present, C_1 indicates the melody is not present, x_i are feature values, and n is the number of real-valued features. The details of features will be discussed in a later paragraph.

$$P(C_0|x_1, \ldots, x_n) \tag{3}$$

By Bayes' theorem, this conditional probability can be rewritten as Equation 4.

$$P(C_0|x_1, \ldots, x_n) = \frac{P(C_0)P(x_1, \ldots, x_n|C_0)}{P(x_1, \ldots, x_n)} \tag{4}$$

With the assumption of independence for each feature, we can rewrite this as Equation 5:

$$P(C_0|x_1, \ldots, x_n) = \frac{1}{Z} P(C_0) \prod_{i=1}^{n} P(x_i|C_0) \tag{5}$$

where Z is:

$$Z = P(x_1, \ldots, x_n) = \sum_{k=0}^{1} (P(C_k) \prod_{i=1}^{n} P(x_i|C_k)) \tag{6}$$

Our features x_i are continuous values. Because we have limited training data, we adopt a Naive Bayes approach and assume they are independent and distributed according

to a Gaussian distribution as in Equation 7. Under this assumption, we can simply collect feature statistics $\mu_{i,k}$ and $\sigma_{i,k}$ from training data to estimate the probability model.

$$P(x_i = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_{i,k}^2}} e^{-\frac{(v-\mu_{i,k})^2}{2\sigma_{i,k}^2}} \quad (7)$$

We now describe the details of features, which are $note\_density$, vel_mean, vel_std, $pitch\_mean$, $pitch\_std$, IOI_mean, and IOI_std:

4.1.1 Note Density

The note density is the sum of all note durations divided by the total length of the music (Equation 8). A melody without rests has a note density of 1, a rest has note density of 0, a sequence of triads without rests has a note density of 3, etc.

$$note\_density = \frac{\Sigma_{note} note.\text{dur}}{total\_length} \quad (8)$$

4.1.2 Velocity

We take the mean and standard deviation of velocity (Equations 9 and 10).

$$vel\_mean = \frac{\Sigma_{i=1}^{N} note_i.\text{vel}}{N} \quad (9)$$

$$vel\_std = \sqrt{\frac{1}{N-1}\Sigma_{i=1}^{N}(note_i.\text{vel} - vel\_mean)^2} \quad (10)$$

4.1.3 Pitch

We take the mean and standard deviation of pitch (Equations 11 and 12).

$$pitch\_mean = \frac{\Sigma_{i=1}^{N} note_i.\text{pitch}}{N} \quad (11)$$

$$pitch\_std = \sqrt{\frac{1}{N-1}\Sigma_{i=1}^{N}(pitch_i.\text{vel} - pitch\_mean)^2} \quad (12)$$

4.1.4 Inter-Onset Interval

The Inter-Onset Interval, or IOI, means the interval between onsets of successive notes. Considering that ornaments and chords may introduce a very short IOIs, we set a window of 75 ms, and when two note onsets are within that window, we treat them as a single onset [13]. IOI calculation is described in detail in Algorithm 1.

4.2 Training data

We compute features for each measure and channel of the training data. For feature selection, we use cross-validation, dividing the training songs into 5 groups, holding out each group and estimating $\mu_{i,k}$ and $\sigma_{i,k}$ from the remaining training data, and evaluating the resulting model by counting the number of measures where the melody channel is judged most likely by the model. We take the average result over all five groups.

Result: The mean and standard deviation of a list of notes in onset-time order
stats is an object that implements the calculation of mean and standard deviation;
note[*i*] is the i^{th} note;
N is the number of notes;
$i \leftarrow 0$;
while $i < N$ **do**
  $j \leftarrow i + 1$;
  while $(j < N) \wedge (note[j].\text{on\_time} - note[j-1].\text{on\_time}) < 0.075$ **do**
    $j \leftarrow j + 1$;
  end
  if $j < N$ **then**
    $IOI \leftarrow note[j].\text{on\_time} - note[i].\text{on\_time}$;
    $stats$.add_point(IOI);
  end
  $i \leftarrow j$;
end
$IOI\_mean \leftarrow stats$.get_mean();
$IOI\_std \leftarrow stats$.get_std();

Algorithm 1: IOI Feature Calculation

After doing this for every combination of features (7 features, thus 127 combinations), for various values of N (the maximum distance to neighboring measures to use in feature calculation), we determine the features that produce the best result for each value of N.

We then re-estimate the probability model using *all* of the training data. In principle, we should also use the training data to learn the best window size N and the best penalty S, but in our training data, melodies are all in one channel, so the ideal value of N for this data should be large, and the ideal S should be zero (highest penalty) to prevent the melody from changing channels. Instead, we will determine N and S from our test data, where every measure is labeled as melody or not, and we will report how these parameters effect accuracy using the test dataset.

4.3 Melodic probability

To prepare for the dynamic programming step, we compute $\boldsymbol{M}_{m,c}$, which is the natural log of the probability of melody in channel c at measure m. (The feature values are different for each combination of m and c.) In the next step, note that if we find the labels with the greatest sum of log probabilities, it is equivalent to finding the labels with the greatest product of probabilities. Logarithms are used to avoid numerical underflow.

4.4 Dynamic Programming

We use dynamic programming to select the channel containing the melody in each measure. Algorithm 2 shows how the assignment of channels maximizes the sum of $\boldsymbol{M}_{m,c}$ values adjusted by subtracting $SP = -\log(S)$ each time the melody changes channels. The backtracking step is not shown since it is standard. [1]

[1] https://en.wikipedia.org/wiki/Viterbi_algorithm

Result: For each measure, determine the channel containing the melody.
N is the number of measures, indexed from 0 to $N-1$;
C is the number of channels, indexed from 0 to $C-1$;
SP is channel switch penalty, a parameter; SP = -ln(S);
$\boldsymbol{A}_{m,c}$ is the accumulated score for measure m and channel c;
$\boldsymbol{B}_{m,c}$ stores the optimal channel number of previous measure;
$\boldsymbol{M}_{m,c}$ tells how melodic is channel c in measure m;
for *i* in [*0* ... *C*) **do**
  $\boldsymbol{A}_{0,i} \leftarrow \boldsymbol{M}_{0,i}$;
end
for *m* in [*0* ... *N*) **do**
  for *c* in [*0* ... *C*) **do**
    $x \leftarrow \boldsymbol{A}_{m-1,c} + \boldsymbol{M}_{m,c}$;
    $\boldsymbol{B}_{m,c} = c$;
    for *i* in [*0* ... *C*) **do**
      $y \leftarrow \boldsymbol{A}_{m-1,i} + \boldsymbol{M}_{m,c} - SP$;
      if $c \neq i \wedge y > x$ **then**
        $x \leftarrow y$;
        $\boldsymbol{B}_{m,c} \leftarrow i$;
      end
    end
    $\boldsymbol{A}_{m,c} \leftarrow x$;
  end
end

Algorithm 2: Dynamic Programming

5. EXPERIMENTS AND RESULTS

5.1 Training

We tried different combinations of features, and the results are shown in Table 1 for windows with 5 measures ($N = 2$). The top 5 feature sets are shown along with the mean and standard deviation of accuracy across 5-fold cross-validation. Differences among the top feature sets are minimal. We use all features except velocity standard deviation.

Table 2 shows the results using each feature individually for 5-measure windows. This shows that all features offer some information (random guessing would be 1/15 or less than 7% correct), but no single feature works nearly as well as the best combination.

If we assume the melody appears in only one channel, which is mostly the case for this training dataset, we can consider the measure-by-measure melody channel results as votes, picking the channel with the majority of votes as the melody channel. Our best feature set (all but velocity standard deviation) gives an accuracy of 96% (2 errors out of 51 songs), using 5-fold cross-validation. In the next section, we relax the assumption that the melody appears in only one channel.

5.2 Testing

Our test dataset labels each measure with a set of channels containing melody. In measures with no melody, this is the empty set. In some measures, the melody is duplicated in different channels, so the label can can contain more than one channel. Note that if we can identify one channel containing the melody, it is simple to search for copies in the other melodies. Our algorithm labels *every* measure with exactly one melody channel. We consider the output to be correct either if it is in the set of true melody channels according to our manual labels, or if the label is the empty set. Typically, the empty set (no melody label) appears in introductions, endings, and measures where the melody channel rests. In these cases (approximately 12% of all measures), there is no clearly correct answer, so we will not include that in the test.

We evaluated accuracy on the test dataset with many values of N and SP. For each value of N, we used the best feature set as determined from the training data and then evaluated the system with different values of SP. The results are shown in Figure 1.

Since N and SP are optimized on the test dataset to obtain a best accuracy of 89.15%, there is some risk of overfitting parameters to the test data. Given more labeled data, we would have used a different dataset to select N and SP, and then we could evaluate the entire system on the test dataset. Instead, we argue that the system is not very sensitive to N or SP, so overfitting is unlikely. Figure 2 shows how accuracy is affected by varying the window size using an optimal value of $SP = 36$ (again, the window includes the measure $\pm N$ measures, so the window size is $2N+1$). This figure shows that 5-measure windows worked the best, but windows up to about 15 measures also work well, with just a few percent variation in accuracy.

Figure 3 shows how the accuracy is affected by varying SP, the switch penalty, using the optimal value of $N = 2$.

nd	pm	ps	im	is	vm	vs	mean	std
1	1	1	1	1	1	0	72.40%	7.20%
1	1	0	1	1	1	0	71.60%	7.06%
1	1	1	1	1	1	1	71.40%	8.26%
1	1	1	1	0	1	0	71.40%	7.70%
1	1	1	1	0	1	1	71.00%	8.83%

Table 1. Mean and standard deviation of accuracy in 5-fold cross validation using the top 5 feature sets, window size = 5. Here, nd means $note\_density$, pm means $pitch\_mean$, ps means $pitch\_std$, im means IOI_mean, is means IOI_std, vm means vel_mean, and vs means vel_std.

nd	pm	ps	im	is	vm	vs	mean	std
1	0	0	0	0	0	0	45.80%	7.22%
0	1	0	0	0	0	0	44.60%	8.11%
0	0	1	0	0	0	0	35.80%	6.50%
0	0	0	1	0	0	0	31.00%	2.55%
0	0	0	0	1	0	0	30.40%	5.64%
0	0	0	0	0	1	0	32.00%	5.87%
0	0	0	0	0	0	1	20.60%	4.10%

Table 2. Mean and standard deviation of accuracy in 5-fold cross validation using individual features, window size = 5

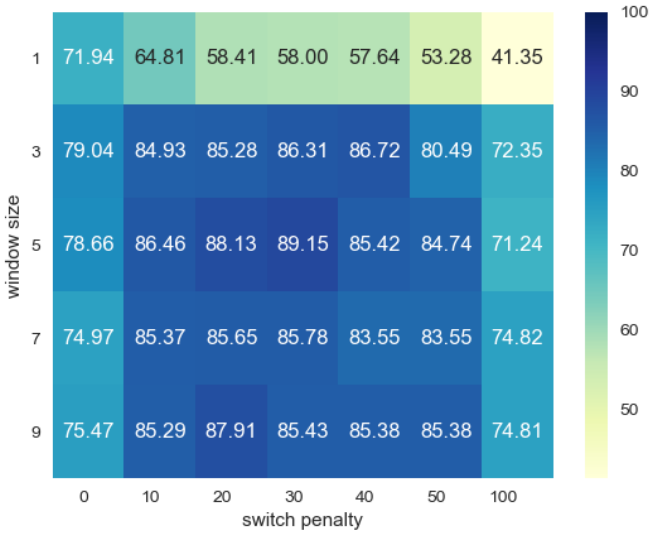

Figure 1. Accuracy for different values of window size and switch penalty.

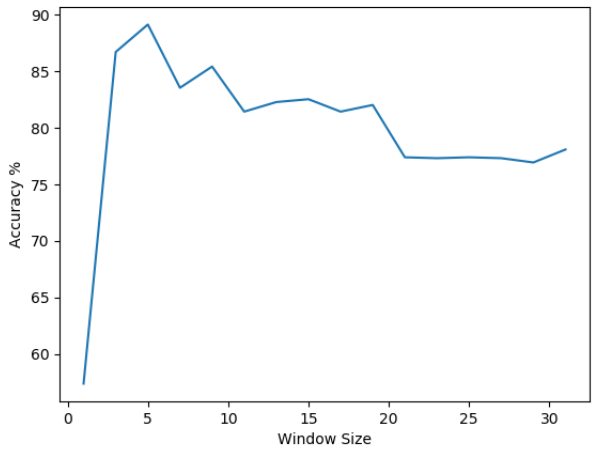

Figure 2. Accuracy on the test dataset vs. Window Size ($= 1 + 2N$) for Switch Penalty = 36.

The best performance is obtained with SP between 30 and 38, but any value from 2 to 38 will achieve performance within a few percent of the best. Since both graphs are fairly flat around the best values of N and SP, the exact values of these parameters are not critical for good performance. In fact, we would expect the best values may depend upon style, genre, and other factors.

From the results, we can observe that the increase of window size helps the performance. The highest accuracy goes from 58.00% to 89.15% when the window size grows from 1 to 5. However, accuracy does not continue to increase for even larger windows. We believe the size of 5 measures is large enough to obtain some meaningful statistical features yet small enough to register when the melody has switched channels. In the next section, we analyze some specific examples of success and failure. We also see that the switch

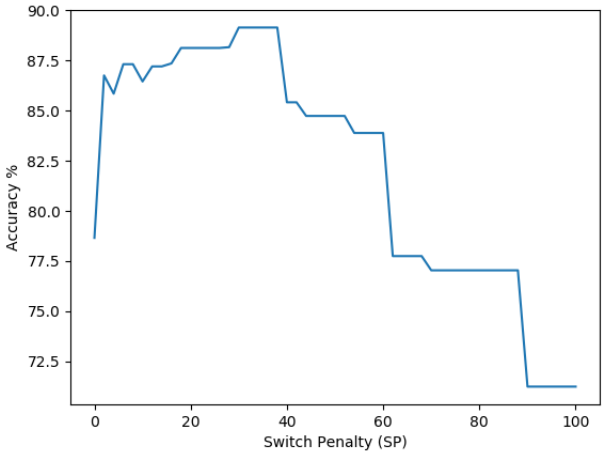

Figure 3. Accuracy on the test dataset vs. Switch Penalty with Window Size = 5. The highest accuracy is 89.15% using any Switch Penalty $\in \{30, 32, \ldots 38\}$.

penalty matters. When we set the penalty to zero, we get the locally best choice of melody channel at each measure, independent of other measures, but it seems clear that this "locally best, indepenent" policy is not particularly good, and this is why we introduced the Viterbi step to our algorithm. On the other hand, when the penalty is very large, we force all measures to be labeled with the same channel. This is not a good policy either, with at best 71.24% accuracy. The results show that our Viterbi step is effective in using context to improve melody identification.

6. ANALYSIS

To better understand our approach, we analyzed some songs in our test dataset.

6.1 A Successful Sample

In most of the cases, this algorithm works well. For example, the figure 4 shows a clip from the popular song "Hotel California." In the figure, the melody is labeled by our algorithm in red (at the top) and other notes are shown in yellow. Notice that this melody is not particularly "melodic" in that it only uses two pitches and there is a lot of repetition. This is one illustration of the need for multiple features and statistical methods. The features for the melody channel have a higher likelihood according to our learned probabilistic model, and the melody is correctly identified.

6.2 Failed Samples

A failure case is shown in Figure 5. Here, the detected melody is shown in red at the top of the figure. The same (red) channel actually contained the melody in the immediately preceding channels, but at this point the melody switched to another channel, shown below in yellow. (In the figure, the lower melody is visually separated for clarity, but both channels actually occupy the same pitch range.) Evidently, the algorithm continued to label the top

Figure 4. The melody detected in the song "Hotel California."

(red) channel as melody to avoid the switch penalty that would be required to label the melody correctly. In fact, the true (yellow) melody appeared earlier in the top (red) channel, so perhaps a higher-level analysis of music structure would also be useful for melody identification and disambiguation.

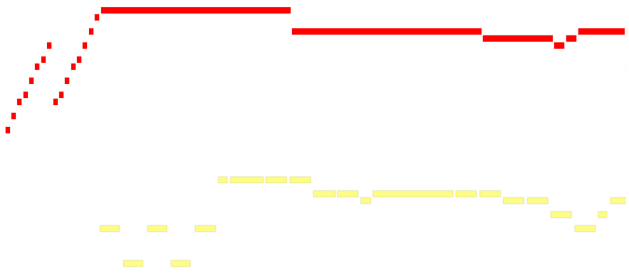

Figure 5. The algorithm identified the top (red) channel as melody of "Being," but the correct melody is shown in yellow at the bottom.

Another song in our dataset is "Ali Mountain." In this piece, at measure 12, the melody is split across two different channels, represented in red (darker) and yellow (lighter) in Figure 6. Taken together, the combined channels would be judged to be very melodic. However, when we consider the channels separately, it is hard to hear whether either is part of a melody, and our algorithm does not rate either channel highly. Since we assume that the melody will be played by one and only one channel within a measure, the melody is not identified in this test case.

Figure 6. Two channels ensemble the melody

7. CONCLUSION

In this paper, we contributed a novel algorithm to detect the melody channel for each measure in a MIDI file. We utilize a Bayesian probability model to estimate the probability that the melody is on a particular channel in each measure. We then use dynamic programming to find the most likely channel for melody in each measure considering that switching channels from measure to measure is unlikely. We obtained an overall accuracy of 89% on our test dataset, which seems to compare favorably to most other results in the literature. The lack of a large shared dataset prohibits a detailed comparison.

Our dataset, including Standard MIDI Files, melody labels, associated software, and documentation are available at the following website:
http://www.cs.cmu.edu/∼music/data/melody-identification.

8. FUTURE WORK

We believe further improvements could be made by studying failures. With bootstrapping techniques, it might be possible to obtain much more training data and learn note-by-note melody identification, which would solve the problem of melodic phrases split across two or more channels. Our current dataset is relatively small, so collecting a larger dataset could be beneficial for tuning this algorithm and developing others. With larger datasets, deep learning and other techniques might be enabled. Perhaps bootstrapping (or semi-supervised learning) techniques could be used starting with the present algorithm to label a larger dataset automatically. We also believe that music structure can play an important role in melody identification. Melodies are likely to be longer sequences that are repeated and/or transposed, and these non-local properties might help to distinguish "true" melodies as perceived by human listeners, even when the melodies are not particularly "melodic" in terms of local features.

Acknowledgments

In this work, we would like to acknowledge Shuqi Dai for providing some test samples in our dataset.

9. REFERENCES

[1] C. Raffel, *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.

[2] B.-M. Thierry, P. E. Daniel, W. Brian, and P. Lamere, "The million song dataset," in *ISMIR 2011: Proc. the 12th InternationalSociety for Music Information Retrieval Conference, October 24–28, 2011, Miami, Florida. University of Miami*, 2011, pp. 591–596.

[3] Z. Wei, L. Xiaoli, and L. Yang, "Extraction and evaluation model for the basic characteristics of midi file music," in *Control and Decision Conference (2014 CCDC), The 26th Chinese*. IEEE, 2014, pp. 2083–2087.

[4] C. Isikhan and G. Ozcan, "A survey of melody extraction techniques for music information retrieval," in *Proceedings of 4th Conference on Interdisciplinary Musicology (SIM08), Thessaloniki, Greece*, 2008.

[5] A. Uitdenbogerd and J. Zobel, "Melodic matching techniques for large music databases," in *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*. ACM, 1999, pp. 57–66.

[6] W. Chai and B. Vercoe, "Melody retrieval on the web," in *Multimedia Computing and Networking 2002*, vol. 4673. International Society for Optics and Photonics, 2001, pp. 226–242.

[7] M.-K. Shan and F.-F. Kuo, "Music style mining and classification by melody," *IEICE TRANSACTIONS on Information and Systems*, vol. 86, no. 3, pp. 655–659, 2003.

[8] L. Li, C. Junwei, W. Lei, and M. Yan, "Melody extraction from polyphonic midi files based on melody similarity," in *Information Science and Engineering, 2008. ISISE'08. International Symposium on*, vol. 2. IEEE, 2008, pp. 232–235.

[9] J. Li, X. Yang, and Q. Chen, "Midi melody extraction based on improved neural network," in *Machine Learning and Cybernetics, 2009 International Conference on*, vol. 2. IEEE, 2009, pp. 1133–1138.

[10] S. Velusamy, B. Thoshkahna, and K. Ramakrishnan, "A novel melody line identification algorithm for polyphonic midi music," in *International Conference on Multimedia Modeling*. Springer, 2007, pp. 248–257.

[11] D. Rizo, P. J. P. De León, C. Pérez-Sancho, A. Pertusa, and J. M. I. Quereda, "A pattern recognition approach for melody track selection in midi files." in *ISMIR*, 2006, pp. 61–66.

[12] H. Zhao and Z. Qin, "Tunerank model for main melody extraction from multi-part musical scores," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2014 Sixth International Conference on*, vol. 2. IEEE, 2014, pp. 176–180.

[13] J. Bloch and R. B. Dannenberg, "Real-time accompaniment of polyphonic keyboard performance," in *Proceedings of the 1985 International Computer Music Conference*, 1985, pp. 279–290.