MOZART OR MESS? HIDDEN MARKOV MODELS FOR MELODY HARMONIZATION

TJ HART, ADAM WARD, MATTHEW WARD

Abstract. In this paper, we explore the process of predicting the unknown chord progression of a piece of music using supervised Hidden Markov Models, a process known as harmonization. The models are trained on data from *melodyhub* [Wu24], an online database containing over 1 million musical scores. We hypothesize that the use of different hidden state space constructions will impact our ability to predict the correct harmonizing chords. Our findings support this conclusion, and we discuss the merits of each construction in maximizing the amount of information available to the model. We also discuss limitations of our modeling, future recommendations, and the ethical considerations present in our project.

1. Problem Statement and Motivation

Have you ever noticed how music seems to have noticeable patterns? The chords change, then repeat themselves in a consistent way, leading to a hummable tune with an almost predictable progression. As music lovers and musicians, we hope to model these patterns using traditional time series analysis methods.

In traditional music (i.e. folk, classical, etc.), the chords and melody are closely interconnected. Often, an experienced musician can listen to a song melody and predict what the harmony would be, i.e. the progression of chords accompanying the melody line. Thus, the melody of a song provides rich information about the underlying chord progression. Since there are a finite number of chords and melody notes used in traditional music, we can model such music as a time series with discrete states.

To do so, we use a Hidden Markov Model (HMM) because its structure matches the format of most music, wherein there is a melody line (observation sequence) accompanied by an underlying harmony line (hidden state sequence) that play simultaneously. Many papers involving music analysis usually employ an RNN structure (such as [CZZR24]), but for our problem and within our limitations, an HMM is most suitable. With this model in hand, our objective is to predict the chords accompanying a given melody, a task we refer to as melody harmonization.

In order to accomplish this task, we faced an uphill battle in converting music into a usable HMM state space. Most music is stored as audio, MIDI,

Date: April 13, 2025.

or standard written music notation, none of which have easy methods to extract melody notes and accompanying chords as time series data. To remedy this, we opt for a less common format called ABC notation [Wik25], which was expressly developed to easily encode single-line melodies and their accompanying chords, and is explained in detail in Section 2. While significant preprocessing is still required to convert music from ABC notation to a state space model, the use of this format helped make our problem tractable. This allowed us to focus on obtaining the results we desired.

2. Data

For this project, we used data from *melodyhub* [Wu24], a database containing over 40,000 monophonic (single melody line) melodies with corresponding chords specifically in the ABC notation format. From the documentation, these songs were complied from the following sources (transformed into ABC notation if needed):

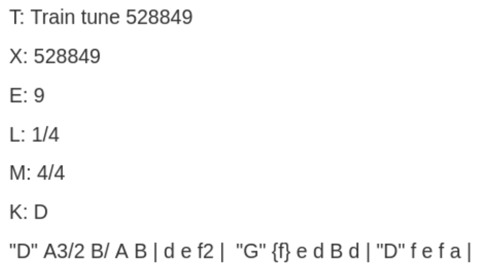
- *JSB Chorales* (4845 songs): Pieces by Johann Sebastian Bach; it is unclear where Melodyhub obtained them.
- *The Session* [Jer] (3044 songs): Primarily traditional Irish music.
- *FolkWiki* [Fol24] (1196 songs): Swedish/Scandinavian folk songs. *Nottingham* [All03] (1014 songs): Contains a variety music styles (waltzs, jigs, and others).
- *ABC Notation* (31612 songs): The vast majority of the music from *melodyhub* comes from a variety of other sources and was not specified (more details can be found here).

ABC notation is a format for storing songs as text that has become fairly standard in the folk and traditional music genres. An ABC file begins with metadata encoding the key, tempo, title, etc. of the song. After the metadata, the song is encoded with letters and symbols to denote melody notes, chords, note duration, and more (see Figure 1a).

As previously mentioned, a large amount of preprocessing was required to construct the HMM structure and prepare the data for training. As such, making extensive use of music21, a package made to parse various notational formats, we performed the following data processing pipeline:

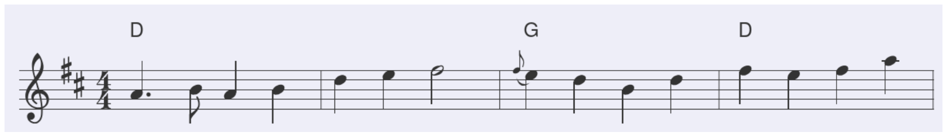
1. Load ABC Songs: Load the song from the raw ABC notation with *music21*. Because some songs had irregular elements that *music21* failed to parse (≈4000 songs, or 1% of the full dataset), we removed these songs from the dataset, to avoid manually working around *music21*'s limitations.

2. Preprocess ABC Songs: Iterate through the elements of the song. We define a next time step of the time series every time a new note or chord is encountered. This is intended to capture the ordered sequence of pitches without the added complexity of note duration. The following information was recorded at each new time step and stored in a *pandas* dataframe with the following columns: measure number (`int`), current beat within the measure (`float`), chord (`int`), melody note MIDI value (`int`).



```
T: Train tune 528849
X: 528849
E: 9
L: 1/4
M: 4/4
K: D
"D" A3/2 B/ A B | d e f2 |  "G" {f} e d B d | "D" f e f a |
```

(A) First line of a song in ABC notation



(B) Corresponding standard musical notation for above song

Figure 1. Example of ABC notation (A) that can be viewed as sheet music (B) using a webapp converter [Wil24].

Now, we sought to study how one chord transitions to another in general. This transition depends almost entirely on the key the song is in. To remove this dependence and make our model generalizable to any key, we performed a 'shift' on the chord and melody note integer values. In particular, for both chord integers or melody MIDI value integers we shifted relative to the current key. For a more in depth view of how the chord and melody note shifting was implemented, see our code attached at the end of this paper.

3. Make Songs into States: Finally, from the song dataframes, generate HMM-ready states and observations, as described in Section 3.

4. Data Split: With preprocessing complete, we performed a randomized train-validation-test split on the dataset. We used a test set of 30% of the data, and the remaining data was split into train and validation datasets of 56% and 14% respectively (an 80-20 split on the remaining 70%). For an overview of some characteristics of the training set songs, see Figure 2.

3. Methods

We begin by formalizing our use of a discrete Hidden Markov Model (HMM) structure. A discrete HMM involves a sequence of hidden states $(X_t)_{t=1}^n \in \mathscr{X}$ with $|\mathscr{X}| = d$, and a corresponding sequence of observations $(Z_t)_{t=1}^n \in \mathscr{Z}$ with $|\mathscr{Z}| = m$. $(X_t)_{t=1}^n$ is a Markov chain, and each observation Z_t depends only on the corresponding hidden state X_t.

Let $F \in \mathbb{R}^{d\times d}$, with

$$F_{ij} = P(X_t = i \mid X_{t-1} = j) \quad \forall t \in \{2, \dots, n\}.$$

We call F the *transition matrix* because it governs the probability of transitioning between any two states.

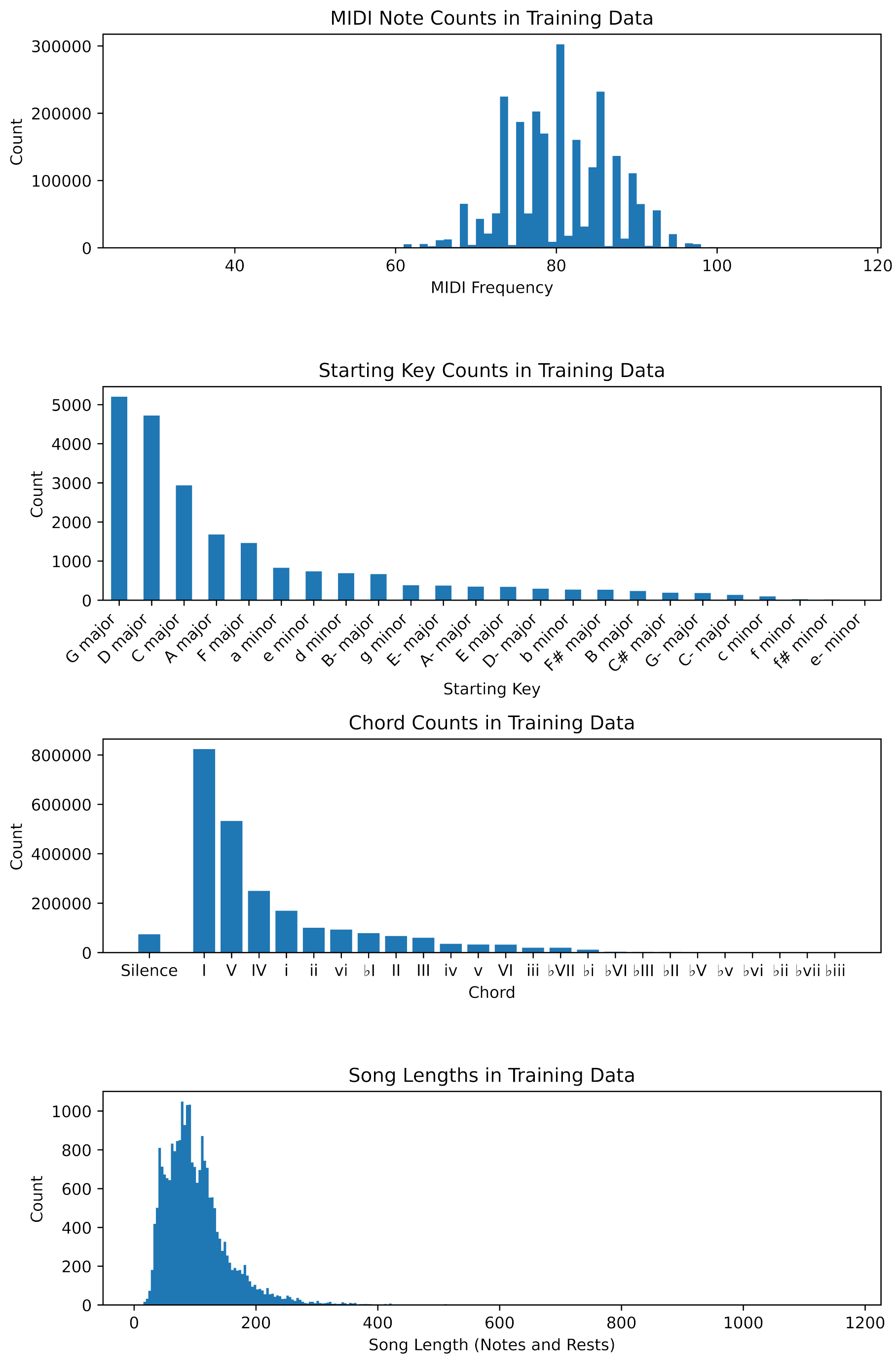


FIGURE 2. Train set summary statistics, from top to bottom: 1) Number of occurrences of each MIDI frequency, with the minimum MIDI value being 28, the maximum being 116. We also counted 57,792 rests in the train set.
2) The number of songs that started in each key. Note that ≈1000 songs had irregular keys,

FIGURE 2 *(continued)*. like dorian or mixolydian, which were removed from the graph for simplicity and readability.
3) Number of chord occurrences represented by Roman numeral (major chords uppercase, minor chords lower case).
4) The distribution of song lengths, with length calculated as total number of notes and rests in the song.

Let $H \in \mathbb{R}^{m \times d}$, with

$$H_{ij} = P(Z_t = i \mid X_t = j) \quad \forall t \in \{1, \ldots, n\}.$$

We call H the *emission probabilities matrix* because it governs the probability of a true hidden state $X_t = j$ "emitting" an observation $Z_t = i$. In other words, it governs the the probability of seeing the observation Z_t given a true hidden state X_t.

3.1. **Creating the HMM.** As described in Section 1, our HMM construction uses an observation sequence $(Z_t)_{t=1}^{n} \in \mathscr{Z}$ representing the melody, and a hidden state sequence $(X_t)_{t=1}^{n} \in \mathscr{X}$ representing information about the harmony (chords), with n being the length of a given song.

Since we intended for our hidden states $(X_t)_{t=1}^{n}$ to encode information we specified, unsupervised methods for training the HMM like the Baum-Welch algorithm would not suffice. Baum-Welch would construct its own optimal hidden states, so instead we created our own transition matrices and emission probability matrices (see 3.2) and used the Viterbi algorithm to find the most likely series of hidden states given the model parameters and a series of new observations.

3.2. **Constructing the transition matrix and emission probabilities matrix.** In order to construct the transition matrix and emission probabilities matrix, we created the dataframe described in Section 2 for each song in the train dataset, then performed the following:

(1) Constructed a list of every unique state in the train dataset and created an index for them.
(2) Constructed a list of every unique observation (melody note) in the train dataset, and created an index for them.
(3) Iterated through the train set songs to count:
    (a) The number of transitions from state $i \in \mathscr{X}$ to state $j \in \mathscr{X}$
    (b) The number of observations of note $i \in \mathscr{Z}$ when in state $j \in \mathscr{X}$
(4) Placed these counts into the transition matrix F and the emission probabilities matrix H respectively, so that F_{ij} is the number of times that songs transitioned to state i from state j, and H_{ij} is the number of times that the melody note i occurred when in state j
    (a) We performed variations on this initialization of F and H in certain cases, as described in Subsection 3.4

(5) Normalized the columns of F and H to be column-stochastic, thus their entries represented probabilities as desired

At the end of this process, we had a transition matrix and emission probabilities matrix with which to initialize a discrete HMM and use the Viterbi algorithm on new observation sequences.

3.3. **Increasing Dimension of Hidden State Space.** For our simplest model, the hidden state space only included a single chord. This matches the rudimentary intuition that a melody note should depend on only the current chord. However, since music also follows chord progressions, as discussed in Section 1, we sought to embed that information into the model. To do so, we added the flexibility to include k chords in the hidden state space ($k \in \mathbb{N}, k \geq 2$), so that a 'running log' of the current chord and past $k-1$ chords was factored into the creation of the transition probability matrix. We also consider including past melody notes in the hidden state space, postulating that including that melody information will allow the model to see patterns in the melody that lead to chord changes.

To make our justification of these ideas more concrete, (and without worrying too much about the musical notation here), consider the following common "One-Four-Six-Five" chord progression, denoted in Roman numerals as: **I**, **V**, **vi**, **IV**, **I**. This progression is one of the most common in all of pop music over the last decade, so it is a good motivating example (see here for a list of popular songs employing this chord progression). If we only consider a single chord in the hidden state space, then the transition matrix will hold probabilities for transitioning from the **I** chord to the **V** chord, the **V** chord to the **vi** chord, etc. However, given that this chord progression is highly common in music, we would expect to find that

$$P(X_n = \mathbf{I} \mid X_{n-1} = \mathbf{IV}, X_{n-2} = \mathbf{vi}, X_{n-3} = \mathbf{V}) > P(X_n = \mathbf{I} \mid X_{n-1} = \mathbf{IV}),$$

i.e. the probability of transitioning from a **IV** chord to a **I** chord would increase if we knew that **V** and **vi** preceded the **IV** chord. Thus, including the past k chords in the hidden state space should allow for more accurate chord predictions. So in practice, we adjust the state space so that $(X_t)_{t=1}^{n}$ takes values in $\mathscr{X}^k$, a space containing "vectors" with k components, each of which are elements of $\mathscr{X}$.

3.4. **Adding a Prior for Transition and Emission Matrices.** Unfortunately, when using larger hidden state configurations (3.3), many observation sequences in the validation set had zero probability under any possible sequence of hidden states. This means both the transition matrix and emission probabilities matrix were sparse, with high probabilities in a few entries and zero probability in others. This led to the following two problems:

(1) For the transition matrix, we note that in many songs there are multiple notes played before the chord changes; it doesn't change every note. Since we created a new state each time either the note or chord changes (see Section 3), this is registered in the state space

as the same repeated chord for each note. Therefore, the diagonal entries of the matrix (i.e. staying in the same state) approached 1, while most other entries approached zero.

(2) Many of the entries in the emission probabilities and transmission probabilities matrices were zero. As a result, given new observation sequences, every possible sequence of hidden states either failed to match the observations or had zero probability of occurring. In this situation, the HMM model jumped to completely unrealistic or nonsensical sequences.

For both of these reasons, our original trained models often failed on the validation set.

To remedy these issues, we added new parameters intended to reduce the diagonal nature of the transition matrix and add a small probability to entries that were previously zero in both matrices.

- λ, which we call the "redact parameter", enforces, during the training described in 3.2, no more than λ repetitions of a chord contributes to the transition matrix. For example, a state sequence such as $[1, 1, 1, 1, 8, 8, 8, 8, 6, 6, 1]$ with $\lambda = 2$ is redacted to become $[1, 1, 8, 8, 6, 6, 1]$ before being trained on. The effect of λ on the transition matrix is shown in Figure 3. It is important to note that λ is only applied when the state space is a single chord.
- `transmat_prior` gives a starting count for each $X_t = i \mid X_{t-1} = j$ with $i, j \in \mathscr{X}$ before training. (See item 3a in 3.2)
- Similarly, `emission_prior` gives a starting count for each $Z_t = i \mid X_t = j$ with $i \in \mathscr{Z}, j \in \mathscr{X}$. (See item 3b in 3.2)

When these parameters are set to zero (or 'None' for λ), the training process in 3.2 is unaffected. We discuss their optimal values in Section 4.

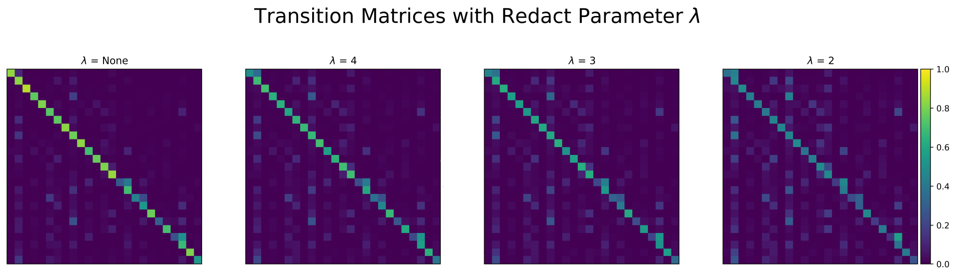


FIGURE 3. Color mappings of the transition matrices. The brighter boxes correspond to probabilities close to 1, while the deep purple boxes correspond to probabilities close to 0. Note how the boxes along the diagonal lose their brightness and while off diagonal boxes get brighter as λ decreases.

3.5. **A New Tool: The Supercomputer.** In 3.3 and 3.4, we introduced a number of tunable parameters, e.g. the number of chords/melody notes in the hidden states, and `transmat_prior` and `emission_prior`. Due to the size of the train set, we found it infeasible to perform a proper grid search on our local machines due to both memory and time constraints.

In order to perform the necessary grid searches over the parameters, we used BYU's supercomputer [oRC24]. By writing a script that created and ran supercomputer jobs, we were able to perform grid searches that would have lasted days on our machines in only a few hours. This required creative use of Slurm [Sch24], a job scheduling system for Linux systems.

4. Results and Analysis

In the following subsections, we describe the results of our modeling and experiments, including the effects of the parameters from 3.4.

Each of these parameters was experimental—it was virtually impossible to guess beforehand their optimal order of magnitude, let alone precise values. Hence, we determined the ranges of the grid searches that follow through initial wider-range searches, which we do not discuss for the sake of brevity.

4.1. **Setting λ (Redact Parameter).** Since our simplest model had only a single chord in the state space, the repeating chords had an especially large effect on the transition matrix, as was shown in Figure 3, which also improved performance. Specifically, when the model was trained with no λ, we obtained an accuracy of 24.30% on the validation set, but when the model was trained with $\lambda = 4$, $\lambda = 3$, and $\lambda = 2$, it achieved an accuracy of 34.51%, 36.60%, and 38.15% respectively. Thus, we saw a 14% increase in accuracy on the validation set through the tuning of this parameter. Running the best model ($\lambda = 2$) on the test set gave a final accuracy of 37.70%, which means that our model generalized well to the test set.

4.2. **Setting `transmat_prior` and `emission_prior` parameters.** In order to find the best values for these parameters, we conducted a grid search over many possible values, then used the validation set accuracies to construct a finer grid search that depends on the dimension of the state space. These grid search values and the accuracies obtained on the validation set are shown in Figure 4, which shows that the optimal parameter configuration for our model is a state space with 1 chord, 0 melody notes, `transmat_prior`$= 50$ and `emission_prior`$= 2025$. Running this model on the test set gave a final accuracy of about 44%.

4.3. **Training different models for major and minor keys.** Since songs in major keys (major songs) can be considered fundamentally different from songs in minor keys (minor songs), we also tried training separate models on only major songs and minor songs. We trained a model on major songs and additionally validated it on major songs in the validation set, doing the same for minor songs. This had mixed results—using the optimal `transmat_prior` and `emission_prior` parameters from above, the major songs model had an accuracy of 48.0%, the minor songs model had an accuracy of 39.6% on their own validation set. For comparison, our best model trained on the full train set had an accuracy of 46.0% on the validation set. Thus, splitting by key may help to some degree, but had mixed results on major vs. minor.

4.4. **Final Analysis.** In the end, our best models performed anywhere between 37% and 44% on the test set. Now, we put a caveat on the accuracies we have obtained. To provide ourselves with a baseline accuracy to compare against, we predicted the **I** chord for the entire validation set, since that chord is the most common among the songs. Therefore, even though our best model achieved 44% on the test set, the improvement from the baseline was not as large as we had hoped, though still promising.

5. Ethical Implications

Thankfully, this project was not fraught with ethical dilemmas. However, there were a few ethical considerations to be made with regard to the dataset used to train our model. As noted in Section 2, the dataset is a grouping of multiple data sources containing different genres of music. The creators of the dataset acknowledge that they "believe all data in this dataset is in the public domain", but that leaves an ethical question as to the validity of this statement, and whether we should use the data if not all of it has been obtained legally (see full statement here).

Now, of the 1,067,747 total songs in the complete dataset, we only used the 41,000 songs labeled for harmonization for our project. While this does not erase the need for caution, our use of about 4% of the total dataset greatly reduces the risk that the songs we used are not in the public domain (assuming that any songs not obtained legally are distributed uniformly throughout the dataset). Thus, we feel that our use of this dataset follows ethical principles to the greatest extent that we are able to verify.

As for the understanding and use of our results, we do not believe that our model could be used in a destructive way to create harm, spread false information, or result in a self-fulfilling loop (other than perhaps predicting looping chords!), unless the model were to be good enough at harmonizing as to take the role of current musicians. As this is not the case, we feel that the use of the model is safe, and that any negative impacts on those who use our model will be negligible.

6. Conclusion

Modeling music effectively with an HMM was much more difficult than we expected at the onset, with the majority of our early effort consisting of the data. Creating a generalizable model that performed better on the validation set than our defined baseline was a challenge, and required us to be creative in our modeling decisions. We found that modifying the transition matrix and emission probabilities matrix using λ, `transmat_prior`,

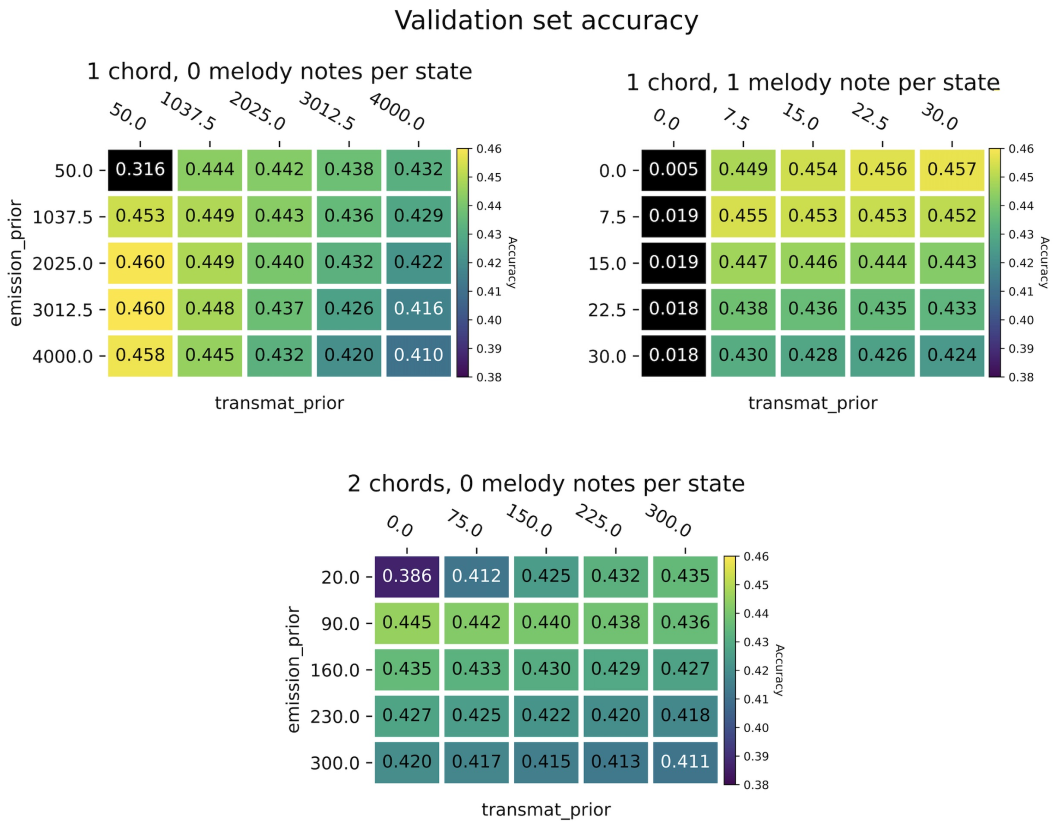


FIGURE 4. Comparing validation set accuracy for different settings of `transmat_prior` and `emission_prior`. The parameters' effect depended heavily on the hidden state space configuration, so we performed a separate grid search for each configuration of chords and melody notes per state. Note the differing axes in each subplot.

and `emission_prior` led to significant improvements in the overall accuracy of the model. Thus, overall we feel proud of what we have accomplished.

In the future, we would consider constructing hidden states in a way that balances the need to carry rich information about the progression of the song with the need to minimize the number of unique states, since increasing the number of unique states in our model drastically increased the computational complexity of the Viterbi algorithm in obtaining our predictions.

Thus, we leave it to the reader to decide: *Mozart or Mess*?

If desired, see our code on GitHub.

References

[All03] James Allwright. Nottingham. https://abc.sourceforge.net/NMD/, 2003. Accessed: 31 March 2025.

[CZZR24] Xinyu Chang, Xiangyu Zhang, Haoruo Zhang, and Yulu Ran. Music emotion prediction using recurrent neural networks, 2024.

[Fol24] Contributors Of FolkWiki. Folkwiki. http://www.folkwiki.se/Meta/Start, 2024. Accessed: 31 March 2025.

[Jer] Jeremy. The session. https://thesession.org/. Accessed: 31 March 2025.

[oRC24] BYU Office of Research Computing. Office of research computing. https://rc.byu.edu/, 2024. Accessed: 31 March 2025.

[Sch24] SchedMD. Slurm workload manager. https://slurm.schedmd.com/documentation.html, 2024. Accessed: 31 March 2025.

[Wik25] Contributors Of Wikipedia. Abc notation. https://en.wikipedia.org/wiki/ABC_notation, 2025. Accessed: 4 February 2025.

[Wil24] Clive Williams. Abc player and editor 2.0. https://abc.rectanglered.com/, 2024. Accessed: 4 February 2025.

[Wu24] Sander-Wood: Shangda Wu. melodyhub. https://huggingface.co/datasets/sander-wood/melodyhub/viewer?views%5B%5D=train, 2024. Accessed: 4 February 2025.