

HON

August 11, 2018

Abstract

1 Trajectory

Here are some thoughts I had about the way we handle trajectories.

```
$ python3 nHON\src\midi_to_trajectory.py <midi_corpus_file> <save_file>
```

1.1 Single Musical Piece

In the earlier implementation we split each musical pieces into multiple trajectories separated by 129. A potential problem with this is that we are forcing independence between the trajectories in the same musical piece.

As an alternative, we can keep a musical piece as one trajectory, and 129 as another “note”. We can then let HON handle the if there are dependencies are not. This is how it is handled in the script `midi_to_trajectory.py`.

1.2 Multiple Musical Piece

In the earlier implementation, different musical pieces are considered as completely separate. When we construct the HON, we construct them separately. This is helpful in understanding the structure of individual musical pieces.

In this approach, to understand the structure of a group of pieces (for example same genre, or same composer), we need to extract features from the multiple HONs and do some form of aggregation.

In the current implementation, different musical pieces are considered as separate trajectories but in the same trajectory file. When we construct the HON the different trajectories in the same file are in the same HON.

So, we have one HON for, say, an genre. If we want to compare different genre we can compare the HONs for these genre as simple graph comparison.

2 Higher Order Network

Let us represent a higher order rule $u_i \rightarrow u_{i+1}$ with priors $u_{i-l}, u_{i-l+1}, \dots, u_{i-1}$ with,

$$u_{i|l} \rightarrow u_{i+1}$$

The *support* of $u_{i|l} \rightarrow u_{i+1}$ is the number of times it is observed in the dataset. And it is represented by $\mathcal{S}(u_{i|l} \rightarrow u_{i+1})$.

The *confidence* of $u_{i|l} \rightarrow u_{i+1}$ is the ration,

$$\mathcal{C}(u_{i|l} \rightarrow u_{i+1}) = \frac{\mathcal{S}(u_{i|l} \rightarrow u_{i+1})}{\mathcal{S}(u_{i|l} \rightarrow *)}$$

where $*$ is any node.

In the current implementation of `hon.py`, there are two parameters – *min_support* (S_m) and *delta_confidence* (C_δ). In the original implementation of *HON*, S_m is the only parameter.

A rule $u_{i|l} \rightarrow u_{i+1}$ is considered for inclusion in the *HON* if

$$\mathcal{S}(u_{i|l} \rightarrow u_{i+1}) \geq S_m$$

A higher order rule $u_{i|l} \rightarrow u_{i+1}$ is included in the *HON* only if the difference in confidence on adding more priors changes the confidence by atleast C_δ .

Suppose $u_{i|l-k} \rightarrow u_{i+1}$ is the longest rule of the form $u_{i|*} \rightarrow u_{i+1}$ that is already included in the *HON*. A higher order rule $u_{i|l} \rightarrow u_{i+1}$ is included if and only if,

$$|\mathcal{C}(u_{i|l} \rightarrow u_{i+1}) - \mathcal{C}(u_{i|l-k} \rightarrow u_{i+1})| \geq C_\delta$$

The confidence of a rule is not equal to the transition probability for $C_m > 1$. So, the transition probabilities are computed separately based on the support after the *HON* has been created.

Run the script as,

```
$ python3 nHON\src\<trajectory_file> <save_file>
```

3 Prediction

Suppose we have multiple *HONs* a trajectory, and we want to find out which *HON* it is most similar to. An example of such task is if we have multiple *HONs* representing musical pieces of different genre, and we have a musical piece that we want to predict the genre of.

In the earlier implementation, we create multiple *HONs* and extract features and approach the problem as a machine learning problem.

In the current approach, we can view the problem as finding the most likely *HON* that generates the trajectory under random walk.

Let us represent the set of *HONs* (also called models) by M , and suppose the trajectory is $t = [u_0, u_1, \dots, u_n]$.

Represent the transition probability of $u_{i|l} \rightarrow u_{i+1}$ in $m \in M$ by $P(u_{i|l} \rightarrow u_{i+1}, m)$.

	Classical (Predicted)	Jazz (Predicted)
Classical (Actual)	420	30
Jazz (Actual)	121	479

Table 1: Results for prediction.

Represent the rule with the longest prior that matches $u_{i|l} \rightarrow u_{i+1}$ in m by $\mathcal{L}(u_{i|l} \rightarrow u_{i+1}, m)$.

Then the log probability that t came from m is given by,

$$\mathcal{P}(t, m) = \sum_{i=0:n} \log(P(\mathcal{L}(u_{i|i} \rightarrow u_n, m), m))$$

The predicted model for t is the one with the max log probability.

$$L(t) = \operatorname{argmax}_{m \in M} \mathcal{P}(t, m)$$

Run the script with

```
$ python3 nHON\src\model_comparison.py <trajectory_file_0> <trajectory_file_1>
...
```

3.1 Experiment

Experiment was performed for classification between Classical and Jazz music. The available trajectories are split into test set and training set randomly with ratio 9 : 1. The *HONs* for the different genre are created using the test trajectories, and the prediction is done with the training trajectories.

The entire experiment was repeated 30 times. The confusion matrix is shown in Table 1.

(Cannot compare directly with the previous approach using ML because we did not do Classical vs Jazz, and I am too lazy to do it again.)

4 Time Duration

One thing we are still missing is the time duration. As discussed, I tried concatenating it with the notes, and adding the durations of consecutive and same notes. That results in networks that are very sparse. So, not sure if thats a good approach.