HON

August 19, 2018

Abstract

1 Trajectory

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

```
$ python3 nHON\src\midi_to_trajectory.py <midi_corupus_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 `HON`s 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 *HON*s for these genre as simple graph comparison.

2 Higher Order Network

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

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

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

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

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

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

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

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

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

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

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 *HON*s 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 *HON*s 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 *HON*s 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 *HON*s (also called models) by M, and suppose the trajectory is $t = [u_0, u_1, \dots, u_n]$.

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

	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 $\mathcal{L}\left(u_{i|l} \rightarrow u_{i+1}, m\right)$.

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

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

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

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

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 *HON*s 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.

5 Centrality

In most of the cases, when we want the centrality values we want them for the nodes – not the node and the combination of nodes and priors.

As an example, assume we have the following probabilities,

$$\begin{aligned} P(a|b \to c) &= p_0 \\ P(b|a \to c) &= p_1 \\ P(a|c \to b) &= p_2 \\ &\vdots \end{aligned}$$

In most applications, the centrality values we want are that of $a, b, c \ldots$ not $(a|b), (b|c), \ldots$. So, computing the centrality values is not a simple case of straight forward application of the centrality functions from the existing network libraries such an networkx.

The most obvious way is to the represent a HON of order m and nodes n as a tensor of dimensions $n \times n \times m$. (But I am not sure if we compute stuffs like eigenvalue etc. easily on tensors. Maybe someone who is more familiar can weigh in here.)

Alternative Approach:

Assume that we have a HON of order m, and we have a rule

$$(u_i|u_{i-1}, \ldots, u_{i-m}) \to u_{i+1}$$

It can be written as,

$$(u_i, u_{i-1}, \ldots, u_{i-m}) \to (u_{i+1}, u_i, u_{i-1}, \ldots, u_{i-m+1})$$

This makes it possible to express the probabilities as a two-dimensional matrix where each rows and columns are of the form $(u_i, u_{i-1}, \ldots, u_{i-m})$. Now that we have a matrix we can use the usual methods.

However this introduces two practical problems:

1. The size of the matrix will be $n^{m+1} \times n^{m+1}$. This is too big for any real application.
2. A high-order node generated on the right may not exist as a left hand in the rules generated. This is because the rule generation process does not accept all possible rules. As a result of this, it will introduce *false* 0 probabilities.

We can solve both these problems in the following ways:

Let $\mathbb{R}$ be the set of all the rules generated. A rule that HON generates is of the form

$$\begin{aligned} r &= (u_i, \ldots, u_{i-m}) \to (u_{i+1}, \ldots, u_{i-m+1}) \\ &= r_L \to r_R \end{aligned}$$

Then let us define the following sets,

$$\begin{aligned} \mathbb{R}_L &= \{r_L : \forall r \in \mathbb{R}\} \\ \mathbb{R}_R &= \{r_R : \forall r \in \mathbb{R}\} \end{aligned}$$

Define a function $\mathcal{L} : \mathbb{R}_R \times \mathbb{R}_L \to \mathbb{Z}$ that matches (consecutively) an element from $\mathbb{R}_R$ to another from $\mathbb{R}_L$ from the left and outputs the length of common match.

Define a function $\mathcal{M} : \mathbb{R}_R \to \mathbb{R}_L$ such that

$$\mathcal{M}(r_R) = \underset{r_L \in \mathbb{R}_L}{\operatorname{argmax}} \, \mathcal{L}(r_R, r_L)$$

If there are sink nodes, it is possible to have the case $\mathcal{M}(r_R) = \phi$. To handle these cases,

$$\mathcal{M}'(r_R) = \begin{cases} \mathcal{M}(r_R) & \text{if } \mathcal{M}(r_R) \neq \phi \\ r_R[0] & \text{if } \mathcal{M}(r_R) = \phi \end{cases}$$

Then we can convert every rule to the form,

$$r = r_L \to \mathcal{M}(r_R)$$

Then, the size of the generated matrix will be approximately of the size $|\mathbb{R}_L|^2$. Since this is much smaller than $n^{2(m+1)}$ and all the r_R are mapped to some r_L, it takes care of both the problems mentioned.

Let us represent the matrix generated by A.

5.1 PageRank

Let I be the image of $\mathcal{M}'(r_R), \forall r_R \in \mathbb{R}_R$.

$$PR(u) = \sum_{r \in I} PR(r) \cdot \delta(u, r)$$

$$\delta(u, r) = \begin{cases} 1 & \text{if } r = r[0] \\ 0 & \text{otherwise} \end{cases}$$

5.2 Closeness Centrality

For $x, y \in I$, let $\rho(x, y)$ be the shortest path between x and y.

For $u, v \in V$, let us define the length of the shortest path as $\xi(u, v)$ as

$$\xi(u) = \min_{\substack{x,y \in I \\ x[0]=u \\ y[0]=v}} |\rho(x, y)|$$

For $u \in V$, the closeness centrality is

$$C(u) = \sum_{v \in V} \frac{1}{\xi(u, v)}$$

5.3 Betweeness Centrality

Let $\iota(u, v)$ be the number of shortest paths between all $x, y \in I$ where $x[0] = u, y[0] = v$.

Let $\iota'(u, v, w)$ be the number of shortest paths between all $x, y \in I$, where $x[0] = u, y[0] = v$ that passes thorough $z \in I$, where $z[0] = w$.

Then the betweeness centrality is,

$$B(u) = \sum_{\substack{s,t \in V \setminus \{u\} \\ s \neq t}} \frac{\iota'(s, t, u)}{\iota(s, t)}$$