Information Technology Course
Module Project
by Damir Dobric / Andreas Pech

Song recommendation using Hierarchical Temporal Memory

Mandar Anil Patkar
mandar.patkar@stud.fra-uas.de

***Abstract*—This paper presents an innovative approach to song recommendation using Hierarchical Temporal Memory (HTM), a cutting-edge technology inspired by the structure and function of the human neocortex. In the context of an increasingly digital and social media-centric world, we aim to enhance user engagement with music playing applications by personalizing song selections to individual preferences. Our methodology leverages neocortexapi to develop a dynamic playlist recommendation system that not only aligns with the user's current interests and tastes but also adapts to changing preferences over time. This system will help to maximize the time users spend interacting with the application, ensuring a seamless and enjoyable user experience. Through a series of experiments and user trials, we demonstrate the effectiveness of our HTM-based recommendation system in providing highly relevant and engaging musical content. The results should indicate a significant improvement in song recommendation and engagement compared to traditional recommendation algorithms. This research not only contributes to the field of music technology but also opens new avenues for applying HTM in other domains where personalized content curation is crucial.**

Keywords—AI, HTM, SDRs, spatial pooler, time-series sequence, multisequence learning.

I. Introduction

In nature there are many events which occur rhythmically, here we are trying to identify such music sequence which is associated entity, and the end goal is to learn song with key and predict the next song in each sequence.

The neocortex is the seat of intelligent thought in the mammalian brain. High level vision, hearing, touch, movement, language, and planning are all performed by the neocortex. Given such a diverse suite of cognitive functions, you might expect the neocortex to implement an equally diverse suite of specialized neural algorithms. This is not the case. The neocortex displays a remarkably uniform pattern of neural circuitry. The biological evidence suggests that the neocortex implements a common set of algorithms to perform many different intelligence functions [1].

The neocortex continually processes an endless stream of rich sensory information. It does this remarkably well, better than any existing AI computer. A wealth of empirical evidence demonstrates that cortical regions represent all information using sparse patterns of activity. To function effectively throughout a lifetime these representations must have tremendous capacity and must be extremely tolerant to noise. However, a detailed theoretical understanding of the capacity and robustness of cortical sparse representations has been missing [2] .

HTM provides a theoretical framework for understanding the neocortex and its many capabilities. To date we have implemented a small subset of this theoretical framework. Over time, more and more of the theory will be implemented. Today we believe we have implemented a sufficient subset of what the neocortex does to be of commercial and scientific value [1].

II. Literature Survey

A. *Sequence*

A particular order in which related things follow each other.

B. *Key*

The sequence which is associated with an event is called key. Here we create the key using the name of playlist with song number as they come in the playlist.

C. *HTM*

Hierarchical Temporal Memory (HTM) provides a flexible and biologically accurate framework for solving prediction, classification, and anomaly detection problems for a broad range of data types. HTM systems require data input in the form of Sparse Distributed Representations (SDRs). SDRs are quite different from standard computer representations, such as ASCII for text, in that meaning is encoded directly into the representation. An SDR consists of a large array of bits of which most are zeros, and a few are ones. Each bit carries some semantic meaning so if two SDRs have more than a few overlapping one-bits, then those two SDRs have similar meanings [3].

HTMs can be viewed as a type of neural network. By definition, any system that tries to model the architectural details of the neocortex is a neural network. However, on its own, the term “neural network” is not very useful because it has been applied to a large variety of systems. HTMs model neurons (called cells when referring to HTM), which are arranged in columns, in layers, in regions, and in a hierarchy. The details matter, and in this regard HTMs are a new form of neural network [1].

D. *Sparse Distributed Representations (SDRs)*

Empirical evidence shows that every region of the neocortex represents information using sparse activity patterns made up of a small percentage of active neurons, with the remaining neurons being inactive. An SDR is a set of binary vectors where a small percentage of 1s represent active neurons, and the 0s represent inactive neurons. The small percentage of 1s, denoted the *sparsity*, varies from less than one percent to several percent. SDRs are the primary data structure used in the neocortex and used everywhere in HTM systems. There is not a single type of SDRs in HTM but distinct types for various purposes [4].

While a bit position in a dense representation like ASCII has no semantic meaning, the bit positions in an SDR represent a particular property. The semantic meaning depends on what the input data represents. Some bits may represent edges or big patches of color; others might correspond to different musical notes. Figure 1 shows a somewhat contrived but illustrative example of an SDR representing parts of a zebra. If we flip a single bit in a vector from a dense representation, the vector may take an entirely different value. In an SDR, nearby bit positions represent similar properties. If we invert a bit, then the description changes but not radically [4].

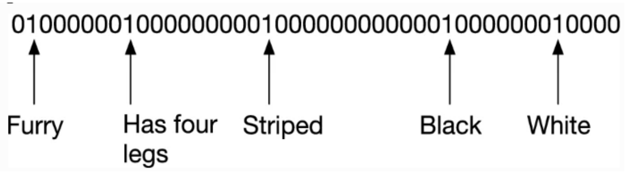


Figure 1 : Encoded data for an animal

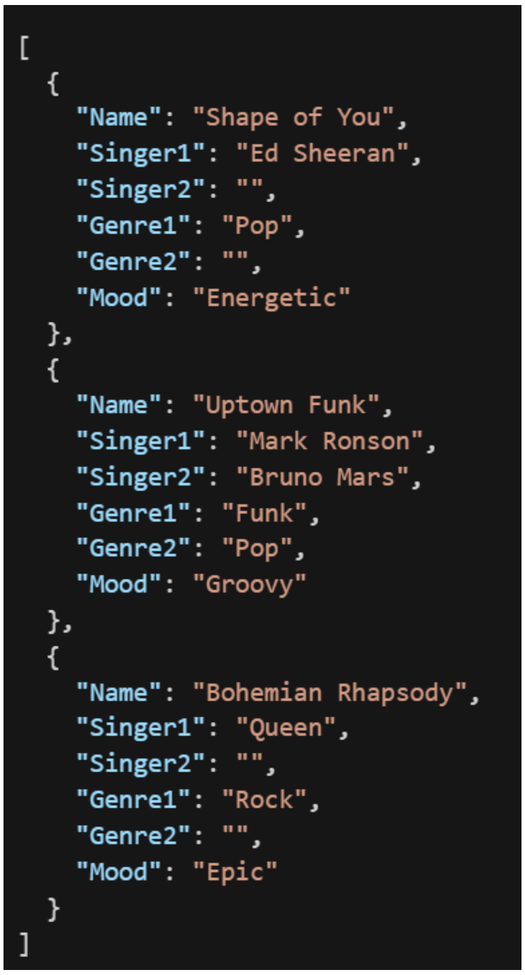
III. METHODOLOGY

In the experiment, the data set for playlists was refined for any error and converted to numeric value. These numeric values are saved in so called class of Database which hold all the attributes of a song. The playlist is then encoded using Scalar Encoder. The encoded data was then trained using algorithm called Multi-sequence Learning. Detailed explanation is given below.

The following methods were used to preprocess, learn and postprocess as part of the experiment.

A. *Read playlist and create database*

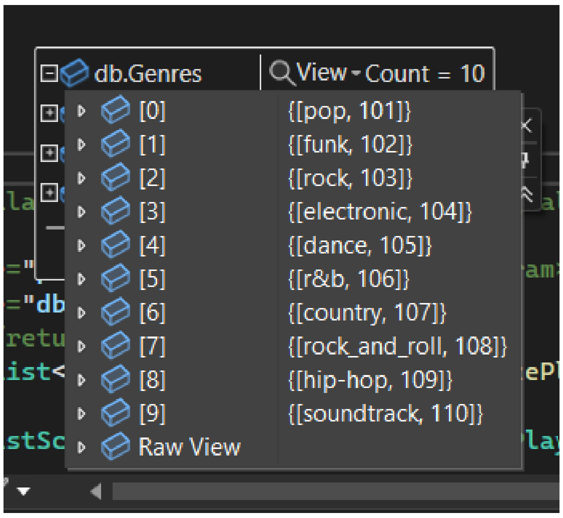
The dataset consists of a list of JSON objects. The objects were of class Song and had its attributes like song name, singer, genre and mood of the song.



```
[
  {
    "Name": "Shape of You",
    "Singer1": "Ed Sheeran",
    "Singer2": "",
    "Genre1": "Pop",
    "Genre2": "",
    "Mood": "Energetic"
  },
  {
    "Name": "Uptown Funk",
    "Singer1": "Mark Ronson",
    "Singer2": "Bruno Mars",
    "Genre1": "Funk",
    "Genre2": "Pop",
    "Mood": "Groovy"
  },
  {
    "Name": "Bohemian Rhapsody",
    "Singer1": "Queen",
    "Singer2": "",
    "Genre1": "Rock",
    "Genre2": "",
    "Mood": "Epic"
  }
]
```

Figure 2 : Dataset

The data has irregularities such as null values and each attribute needs to be mapped to a unique value. Figure 3 show the transformation from raw data to attributes of say genre having mapped to a unique numeric value.



```
db.Genres    View  Count = 10
  [0]        {[pop, 101]}
  [1]        {[funk, 102]}
  [2]        {[rock, 103]}
  [3]        {[electronic, 104]}
  [4]        {[dance, 105]}
  [5]        {[r&b, 106]}
  [6]        {[country, 107]}
  [7]        {[rock_and_roll, 108]}
  [8]        {[hip-hop, 109]}
  [9]        {[soundtrack, 110]}
  Raw View
```

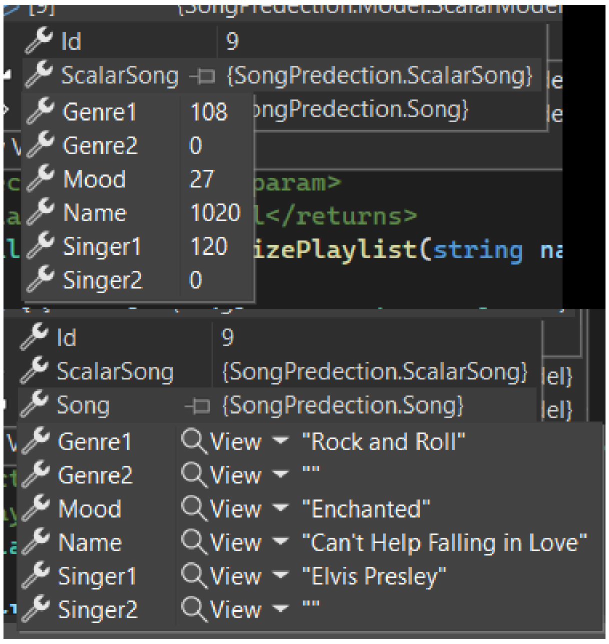
Figure 3 : Unique values assigned to genre in playlist

The initial values for all the attributes of songs are predefined in static variables. So, if it's a three-digit value we could accommodate 999 unique entries in database for the selected attribute.

In short, a database is a class holding dictionary of all the attributes of a song. Each attribute has a unique entry holding key as name of attribute and value as numeric value assigned to it. Database is a very important class which is used later in scalarizing, configuring encoder, determining size of SDR and decoding the predictions.

B. *Scalarize Playlist*

For the experiment, the class of Database holds all the actual value which are seen in dataset and mapped scalar values.



```
Id          9
ScalarSong  {SongPredection.ScalarSong}
  Genre1    108
  Genre2    0
  Mood      27
  Name      1020
  Singer1   120
  Singer2   0
```

```
Id          9
ScalarSong  {SongPredection.ScalarSong}
Song        {SongPredection.Song}
  Genre1    View  "Rock and Roll"
  Genre2    View  ""
  Mood      View  "Enchanted"
  Name      View  "Can't Help Falling in Love"
  Singer1   View  "Elvis Presley"
  Singer2   View  ""
```

Figure 4 : Scalarize song v/s actual song attributes

As shown in Figure 4, the song is the smallest level of data structure which can used and manipulated. All the values from class Song are scalarized and stored in ScalarSong and if found any null values they are replace by 0 representing that it does not exists.

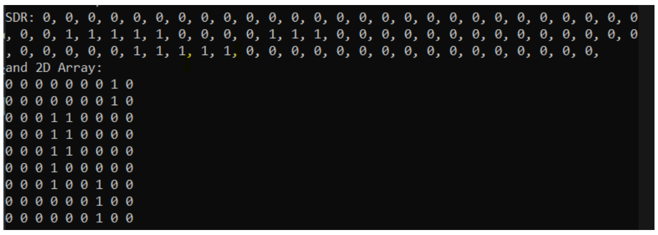
Looking in Figure 3 where all the genres are assigned unique value and Figure 4 the Genre1 is "*Rock and Roll*" so the ScalarSong has scalar value of Genre1 as "*108*" if compared back in Figure 3 and so on the for all the playlists and all the attributes of song is done the same way. Thus, we obtain ScalarSong with one-to-one mapping saved for easy retrieval later.

C. *Encode playlist*

Encoding is done using Scalar Encoder of NeoCortexApi and the output is SDR. But before we start encoding, we need to configure the Scalar Encoder and understand how the multiple attributes of the songs come together to form and complete SDR.

The main features required by the Scalar Encoder are the size which will represent the attribute, minimum scalar value of the attribute and the maximum scalar value. All these values are dynamically set as per our dataset and usually calculated while creating class of Database. There are more features in the encoder but are of the least significant during our experiment.

Following is the encoded output for a random song during running the experiment in Figure 5.



```
SDR: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
and 2D Array:
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
```

Figure 5 : SDR of encoded attribute of song

Now that we have configured the encoder, the SDRs are concatenated as Singer1, Genre1 and Mood of the song in the respective sequence.

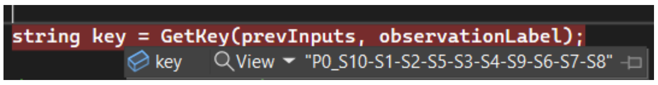
D. *Multisequence Algorithm*

The Multisequence Learning is based on HTM Classifier taken from the NeoCortexApi. In this algorithm, the connections of taken as per HTM Configuration. These configurations are responsible for the mapping to actual cortex of human brain. Spatial Pooler is used to create a sparse representation of encoded data and Temporal Memory is used to remember the sparse representation. There is also a Cortex Layer and HTM Classifier which gets trained model and predict the label.

Following is the algorithm for Multisequence Algorithm.

```
01. Get HTM Config and initialize memory of Connections
02. Initialize HTM Classifier and Cortex Layer
03. Initialize HomeostaticPlasticityController
04. Initialize memory for Spatial Pooler and Temporal Memory
05. Add Spatial Pooler memory to Cortex Layer
        05.01 Compute the SDR of all encoded segment for multi-sequences using Spatial Pooler
        05.02 Continue for maximum number of cycles
06. Add Temporal Memory to Cortex Layer
        06.01 Compute the SDR as Compute Cycle and get Active Cells
        06.02 Learn the Key with Active Cells
        06.03 Get the input predicted values and update the last predicted value depending upon the similarity
        06.04 Reset the Temporal Memory
        06.05 Continue all above steps for sequences of multi-sequences for maximum cycles
07. Get the trained Cortex Layer and HTM Classifier
```

The Key which is learn with active cells is most important piece which is remember while learning happens. The has name of the playlist and ID of the songs which is generated during the scalarizing of playlist is done. Thus it looks as name of playlist plus a chain of IDs of song in order which they appear in a playlist. Figure 6 is for better visualization of the Key.



```
string key = GetKey(prevInputs, observationLabel);
         key   View  "P0_S10-S1-S2-S5-S3-S4-S9-S6-S7-S8"
```

Figure 6 : Key during one of the cycles while running experiments

E. Prediction using model and decoding prediction

The object of Cortex Layer is to compute the SDR (Sparse Distributed Representation). The object of HTM Classifier is to predict possible value.

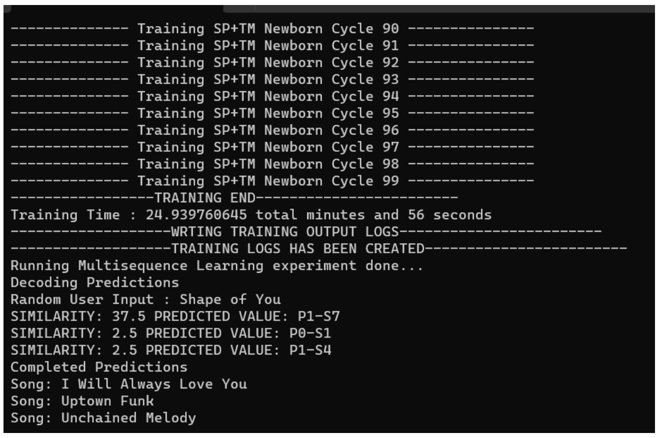
The predicted out put is Key and needs to be interpreted as per the inputs which were given so a decoding is necessary before publishing results. The predicted key holds the playlist number and the from the chain of songs the last appearing song should be picked up as that's the song which appears at the end of playlist, and it also matches the key when the algorithm learned the relationship between key and active cells.

IV. RESULTS

In this experiment, Key created from the song in a playlist holds significance important for learning the sequence. So, each playlist is given a unique name and each song hold a unique entry in *Database*. The key is created by as *'{Playlist.Name}-{Song1}-{Song2}-{Song3}-{Song4} and so on'* adding the song at end of key and learn with a SDR of a last song in the key.

Once the learning part is done, a song is selected at random from a random playlist and prediction is executed. On the result of the prediction, the key is mapped back after looking in *Database* and songs are displayed. Multisequence Learning algorithm has theoretically 100% accuracy [6].

Figure 7 shows the final output from one of runs of experiment.



```
-------------- Training SP+TM Newborn Cycle 90 ---------------
-------------- Training SP+TM Newborn Cycle 91 ---------------
-------------- Training SP+TM Newborn Cycle 92 ---------------
-------------- Training SP+TM Newborn Cycle 93 ---------------
-------------- Training SP+TM Newborn Cycle 94 ---------------
-------------- Training SP+TM Newborn Cycle 95 ---------------
-------------- Training SP+TM Newborn Cycle 96 ---------------
-------------- Training SP+TM Newborn Cycle 97 ---------------
-------------- Training SP+TM Newborn Cycle 98 ---------------
-------------- Training SP+TM Newborn Cycle 99 ---------------
-----------------TRAINING END------------------------
Training Time : 24.939760645 total minutes and 56 seconds
-------------------WRTING TRAINING OUTPUT LOGS------------------------
-------------------TRAINING LOGS HAS BEEN CREATED------------------------
Running Multisequence Learning experiment done...
Decoding Predictions
Random User Input : Shape of You
SIMILARITY: 37.5 PREDICTED VALUE: P1-S7
SIMILARITY: 2.5 PREDICTED VALUE: P0-S1
SIMILARITY: 2.5 PREDICTED VALUE: P1-S4
Completed Predictions
Song: I Will Always Love You
Song: Uptown Funk
Song: Unchained Melody
```

Figure 7 : Random user input and predicted output

V. DISCUSSIONS

Learning and predicting sequences have been experimented with for decades and various methods have been used. Adapting an existing method for large and noisy data remains a challenge. In the experiment, HTM Classifier is used which is a recently developed neural network based on cortex of human brain and not just a single neuron model.

Following are couple of improvements which can be done to further refine the experiments:

A. Consider null values while learning

When we consider null values for each attribute, we add one more tuple and this can be filled when we do not know the value, or if the value is null.

For example: Usually a song has a primary singer and secondary singer but not all songs have a known singer, or the artist might be unknown. So, in such scenarios we can assign a value which adds one more tuple and increases some meaning while learning for unknown attributes.

B. Consider all significant possible parameters for inputs

This improvement comes from the previous improvement where null values should be filled with data; this helps to use new attributes while learning.

For example: Usually a song has a primary singer and secondary singer but not all songs have them, so in such cases we can have secondary singer a null and consider singer 2 as attribute while learning.

C. Create larger dataset to improve input sequences

The size of the dataset has always been a significant challenge to deal with. There are a couple of ways to tackle it.

1) Synthetic data

One can create a playlist by randomly choosing songs from a large data set of songs. The downside to this is that the learning algorithm ends up learning the randomness of data instead of real-world relationship between two songs.

2) Scrapping web data

This one of the easy methods where web scrappers can be used to scrape the playlist from music related web application hosted online. To make it even quicker, one can request playlists from generative AI.

3) Data from user

This method is most genuine method and most time consuming too. Here we need to gather data from real-time users where the user creates their own playlist in application, and we add that data while continuous learning.

VI. REFERENCES

[1] J. Hawkins, "numenta.com," Numenta, 12 September 2011. [Online]. Available: https://numenta.com/neuroscience-research/research-publications/papers/hierarchical-temporal-memory-white-paper/. [Accessed 22 March 2022].

[2] S. A. &. J. Hawkins, "arxiv.org," Arxiv, 25 March 2015. [Online]. Available: https://arxiv.org/abs/1503.07469. [Accessed 22 March

2022].

[3] S. purdy, "arxiv.org," Arxiv, 18 February 2016. [Online]. Available: https://arxiv.org/abs/1602.05925. [Accessed 22 March 2022].

[4] K. j. H. &. S. Ahmad, "link.springer.com," SpringerLink, 20 July 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42452-021-04715-0#Sec14. [Accessed 22 March 2022].

[5] "Dataset: https://github.com/numenta/nupic/blob/master/src/nupic/datafiles/extra/hotgym/rec-center-hourly.csv".

[6] "NeoCortexApi : https://github.com/ddobric/neocortexapi".