Implement anomaly detection sample

Md Zahid Hasan
md.hasan5@stud.fra-uas.de

Md Mosharraf Hossain
md.hossain3@stud.fra-uas.de

***Abstract*— A machine learning algorithm approach inspired by biology both structurally and functionally is called Hierarchical Temporal Memory (HTM) it's like the neocortex of a human brain. It employs a hierarchical arrangement of nodes to process time series data in a decentralized manner. According to, each node and column, it can train to learn and recognize the patterns of the input data set. This capability enables the processing of information, recognition, and identification of patterns, and the making of future predictions based on previous learning. In this contemporary era, this process has been hugely used for anomaly detection and prediction in numerous sectors such as geological disasters, cyber-intrusion detection, military surveillance, system fault/damage detection. In our Project, we have implemented an anomaly detection sample through the Hierarchical Temporal Memory where we had multiple double sequences JSON data, and based on this, HTM model learns training and predicting data. After that, the model can detect anomalies within the predicted data according to the tolerance value which is, given by the user. We also generate a graph which delineates anomalies.**

Keywords—Anomaly Detection, HTM, Multisequence Learning, Neocortex API, Machine Learning.

I. INTRODUCTION

Numenta has developed Hierarchical Temporal Memory (HTM), a machine intelligence technology with biological constraints. The brain scientist Jeff Hawkins, the founder of the Redwood Neuroscience Research Institute and Sandra Blakeslee, published it in 2004 [1]. It is a Machine Learning approach based on the Thousand Brains Theory. This technology operates according to a theory about how the biological neocortex works. Mammals only have one area of the brain called the neocortex, which is responsible for higher order processes like language, conscious movement and thought, and sensory perception [2]. We can think of HTM as a particular type of hierarchical Bayesian model. In order to learn the structure and invariance of the space of problems, it also makes use of spatial-temporal theory [3]. In recent years HTM also used for anomaly detection. Something that differs from the expected standard or normal state is called an anomaly. The terms outliers, discordant observations, exceptions, aberrations, surprises, etc. are frequently used to describe the anomalies. Finding anomalous patterns in data is known as anomaly detection. Anomaly detection has been extensively researched and used in many significant domains in recent years, including early warning systems for geological disasters, cyber-intrusion detection, military surveillance, and system fault/damage detection. As related fields develop, more applications in emerging industries like insurance, healthcare, and distributed sensor network surveillance begin to attract attention [4].

By identifying complex temporal patterns and relationships in data, HTM aims to mimic the fundamental workings of the neocortex and predict future events. This methodology excels in simulating sequential learning and related approaches like recurrent neural networks' (RNNs) Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM).

A machine learning model called the Hierarchical Temporal Memory Cortical Learning Algorithm (HTM CLA) is intended to replicate the information processing processes of the neocortex in the human brain. Several essential elements are included in this algorithm to handle input data. An encoder first encodes the raw input data, converting it into a sparse distributed representation (SDR). After being run through a spatial pooler, this SDR—which contains information in binary format with few active bits—becomes more resilient to noise. After that, the output is processed by the Temporal Memory component, which is in charge of identifying and picking up on patterns in the data. These acquired patterns are used by the Classifier component to categorize input data and forecast new patterns. Furthermore, the Homeostatic Plasticity Controller guarantees that the system continuously learns new patterns over time [5].

II. METHODOLOGY

To detect Anomalies in our project we used the Neocortex API, which was developed within the .NET framework, we also used Hierarchical Temporal Memory (HTM) for its functionality because of its best functionality. In our project testing and predictions are carried out through real numerical data which we will discuss elaborately in the below, which was taken from the Numenta Anomaly Benchmark (NAB) [6] Twitter_volume_GOOG dataset, which we stored in individual JSON files, We have used JSON over other formats like CSV or XML because it can handle more complex and large amount of data, adhering to the JSON format throughout the project.

A. Data Set 1:

For this project, our primary data source was the Numenta Anomaly Benchmark (NAB) [6], a repository commonly used for evaluating anomaly detection algorithms. Specifically, we focused on extracting real-time tweet counts of Google per hour, encapsulated as numerical sequences provided by NAB, we convert these numerical sequences into JSON files.

In delineating our dataset, we allocated 32 hours for analysis, partitioned into 16-hour segments for training and predicting. The training phase encompassed data spanning from 12:00 am on March 1, 2015, through 4:00 pm on the same day. Subsequently, the predicting phase involved data from 12:00 am on March 2, 2015, to 4:00 pm on the same day. Our dataset adheres to a structured format, with distinct folders containing training and predicting data. Within the "training_files" directory, four JSON files contain the requisite data for training purposes. Similarly, the "predicting_files" directory hosts an equivalent set of four JSON files suitable for predicting future trends.

Below, we provide insight into the structure and content of our data sequences, summarize within these JSON files. This systematic arrangement ensures clarity and accessibility throughout our analysis and model development efforts. For example, an hourly sequence has a list of 12 numerical values per hour: [14, 14, 9, 13, 7, 7, 5, 13, 11, 7, 9, 9]. Our JSON structure is like the data given below:

Listing 1 dataset 1 (NAB) structure

```
{
  "sequences": [
          [13, 16, 9, 5, 10, 7, 5, 9, 13, 14, 9, 8],
          [12, 4, 11, 11, 11, 18, 6, 5, 8, 9, 5, 15],
          [13, 6, 14, 7, 1, 8, 9, 5, 6, 10, 6, 6],
          [10, 16, 8, 9, 5, 16, 6, 10, 11, 6, 16, 14]
                    ]
}
```

B. *Dataset 2:*

We also use another kind of dataset as JSON files (which is fabricated) to predict anomaly detection, where the full process is like the above dataset, we did this to test our HTM process and the accuracy level test for various types of data. The fabricated dataset is based on the temperature of Bangladesh in the year 2023 [7]. Firstly, we keep both training and predicting datasets in output folder in the repository inside this folder, used datasets folders we can use the datasets when we need, we take them from these folders and insert the data into the training_files and predicting_files folders.

Here are also 4 files each file has 5 sequences where each sequence has 10 numerical data (means 10 days temperature value) for training, which is located in the training_files folder, and also 4 files each file has 5 sequences where each sequence has 10 numerical data (means 10 days temperature value) for predicting which is located in the predicting_files folder. We took a total of 400 days of data. Our Dataset structure is like the data given below:

Listing 2 dataset 2 (fabricated) structure

```
{
"sequences": [
[35, 26, 38, 28, 33, 27, 32, 29, 36, 25],
[34, 26, 33, 21, 37, 26, 31, 25, 39, 31],
[37, 29, 36, 28, 39, 33, 32, 25, 33, 27],
[23, 17, 24, 16, 22, 15, 23, 14, 21, 14],
[18, 14, 22, 15, 17, 13, 22, 15, 24, 17]
                ]
}
```

C. *HTM configuration:*

The process starts with HTM Capturing Configuration and initializing the connection memory then executes the Spatial Pooler and Temporal Memory after that Spatial Pooler Memory is added to the cortex layer and trained for a maximum number of cycles. Since completing the full cycle of numbers, the trained cortex layer and the HTM classifier are returned.

The class requires the configuration settings for both the Encoder and HTM components to be passed to the next level. Then we will utilize the classifier object derived from the trained HTM to make predictions, aiding in anomaly detection.

Currently, we are in the process of preparing training and predicting datasets, which will encompass integer values ranging from 0 to 100. Importantly, these datasets lack any periodic patterns. To facilitate this task effectively, we are adhering to specific settings outlined in a provided listing.

In this configuration, we've chosen to express these integer values using 21 active bits. This decision aims to achieve a balance between precision and efficiency, considering that we're working with 101 distinct values ranging from 0 to 100.

To ensure our encoding scheme effectively covers the entire spectrum of integer values in this dataset, it's essential to compute the total number of input bits needed. This computation, designated as "n," involves adding the count of buckets (101, representing the integer values) to the width of the representation (21 for the active bits), then subtracting one. Thus, the calculation results in n = buckets + w – 1 = 101 + 21 – 1 = 121 [8].

By understanding and implementing these settings, we can effectively encode our data for subsequent training and predicting within our project, ensuring that the representation adequately captures the nuances of the dataset within the specified integer range.

Listing 3 Parameter values set for Encoder

```
int inputBits = 121;
int numColumns = 1210;
Dictionary<string, object> settings = new
Dictionary<string, object>()
        {
          { "W", 21},
          { "N", inputBits}
        };
```

The lower and upper bounds of the data are configured to 0 and 100, respectively, under the assumption that all values will fall within this range. It is important to note that these bounds may need adjustment based on the specific input data being used. We have made some changes to HTM encoder's parameters which is given in above code in Listing 3. We change this because of we faced various issues for this when we run the program it does not works properly and shows some error also then we testing through various values. At last we found these values 121 and 1210 for inputBits and numColumns respectively which works properly for our project [Listing 3].

D. Code Execution Process:

In our project, we have two types of datasets so first of all you have to decide which data you want to keep for execute this project. We keep the NAB dataset as primary data inside the folder, “training_files”, and “predicting_files”, based on our project directory. To execute our project, clone the project first, then open the terminal write dotnet run and press enter then the system asks for a tolerance value from you so give a tolerance value like 0.2 (20 %). Based on this tolerance value our system predicts anomalies for example: if the predicted value is more than 20% then the system declares anomalies.

Initially, we generate separate JSON files for training and prediction data, organizing them into distinct folders within our main project directory. Subsequently, we proceed to read data from these folders and train our HTM model in this stage for reading purposes we create (jsonfileread) method to read the files from the folder [Listing 4].

Listing 4 Reading files from the folder

```
public JsonFolderReader(string folderPath)
    {
        AllSequences = new List<SequencesContainer>();

            // Get all JSON files in the specified folder

        string[] jsonFiles = Directory.GetFiles(folderPath,
                                    "*.json");
        ............................................................
        ………………………………………....
```

We extract the numerical sequences from JSON files present inside both the training and predicting folders and use them to train the HTM model using multisequence-learning. In our project, we implemented a class that is “MultiSequenceLearning” where we use the Neocortex API functionalities which help us train and predict using hierarchical temporal memory [Listing 5].

Listing 5 Collecting paths for Training and predicting folders

```
// Get the solution directory path
string solutionDirectory =
Directory.GetParent(AppDomain.CurrentDomain.Bas
eDirectory).Parent.Parent.Parent.FullName;

// Construct paths for training and predicting
Folders
string trainingfolderPath =
Path.Combine(solutionDirectory, "training_files");
string predictingfolderPath =
Path.Combine(solutionDirectory, "predicting_files");
```

After extracting data from JSON files, then we converted the extracted data into a suitable format for HTM training [Listing 6].

Listing 6 Converting sequences for HTM input

```
foreach (var sequencesContainer in
sequencesContainers)
  {
        var sequences = sequencesContainer.Sequences;

            foreach (var sequence in sequences)
                        {
List<double> convertedSequence = sequence.Select(x
                   => (double)x).ToList();

            string sequenceKey = "S" + sequenceIndex;
mysequences.Add(sequenceKey, convertedSequence);
                    sequenceIndex++;
                        }
```

After that we use multisequencelearning classes for training our HTM model which is given below. Where we used the predictor object as a trained HTM model to predict anomalies from our extracted data from predicting_files [Listing 7].

Listing 7 Train the model using MultiSequenceLearning

```
    // Train the model using MultiSequenceLearning
MultiSequenceLearning myexperiment = new
MultiSequenceLearning();
      var predictor = myexperiment.Run(mysequences);
                  predictor.Reset();
```

After that we use list of lists to store numerical sequences which we are using for anomaly detection, and anomaly indices: indices where we can find anomalies. This is important for plotting. More about this later [Listing 8].

Listing 8 Store data to list of lists

```
  // Create lists to store all data and anomaly indices
   List<double[]> allData = new List<double[]>();
   List<List<int>> allAnomalyIndices = new
List<List<int>>();
```

After that we use the `AnomalyDetectMethod´ method of the AnomalyDetection class, we pass the numerical sequences one by one to our predictor model to detect anomalies. Please note that before passing list of numerical sequences, we are trimming a few values in the beginning randomly (such as two or three values) to get more accuracy in our model [Listing 9].

Listing 9 Convert the sequences & Trimming some values

```
// Convert the sequence to a list of double values
List<double> inputlist = sequence.Select(x =>
(double)x).ToList();
 double[] inputArray = inputlist.ToArray();
 // Trim some values randomly in the beginning of the
sequence
            Random random = new Random();
            int trimCount = random.Next(1, 4);
             double[] inputTestArray =
             inputArray.Skip(trimCount).ToArray();
// Get the anomaly indices from the
AnomalyDetection class
                  List<int> anomalyIndices =
AnomalyDetection.AnomalyDetectMethod(predictor,
inputTestArray, tValue);
```

In the end, we are going to use AnomalyPlotter class under PlotAnomaly.cs to plot our anomalies [Listing 10].

Listing 10 Plot anomalies graph

```
AnomalyPlotter.PlotGraphWithAnomalies(allDa
ta, allAnomalyIndices);
```

In our system, we are going to iterate through each value of a numerical sequence which is passed through the inputTestArray parameter to the "AnomalyDetectMethod" method. The trained model output: predictor is used to predict the next element for comparison. We use an anomalyscore ratio to calculate and compare to detect anomalies if the prediction crosses a certain tolerance level, it is taken as an anomaly. We can pass the tolerance value from outside to the method mentioned above [Listing 11].

Listing 11 Predictor to predict Anomalies

```
var res = predictor.Predict(item);
```

After that, the prediction derived from the predictor model is in the format of, "NeoCortexApi.Classifiers.ClassifierResult`1[System.String]". We use string operations to extract data from it [Listing 12].

Listing 12 String Operation to extract data

```
var value1 =
res.First().PredictedInput.ToString().Split('-');
var value2 = res.First().Similarity;
```

In our project, generally we get the output from the HTM is in the following format in our project, when we pass a numerical value 14 for example [Listing 13].

Listing 13 Output looks like given below

```
S3_11-5-12-10-14-13 - 100
S1_5-6-16-10-4-11-7 - 5
.....
```

The first line has the best prediction which the HTM model predicts, with accuracy. We can easily derive the predicted value, which will come after 14 (in this case, it is 13). The string operations are used to get these values. Later we are going to use this to determine anomalies [Listing 14].

Listing 14 Process of anomalies determination

```
        .............
                    int nextIndex = i + 1;
                    double nextItem = list[nextIndex];
                    double predictedNextItem =
double.Parse(value1.Last());

  // Calculating the anomaly score and deviation
                    var AnomalyScore =
Math.Abs(predictedNextItem - nextItem);
                    var deviation = AnomalyScore /
nextItem;
        ..........
```

In our project, we used 'AnomalyScore', which is nothing, but the absolute value of the ratio of differences between HTM´s predicted number and actual number. If the ratio exceeds tolerancevalue, we mark it as an anomaly, otherwise, it is not. When an anomaly is detected, we skip that element in the list (we did not pass that value to HTM in the loop).
We use accuracyPerList to record accuracy per numerical sequence tested. recordAccuracy is collected from inside each loop run, which indicates HTM model´s accuracy. We also add index positions of anomalies to anomalyIndices, when we encounter indices of anomalies in loop. totalAccuracy and listCount is used to calculate average accuracy of the whole experiment.

After that we use static variables to access these from outside, i.e: in Program.cs [Listing 15].

Listing 15 Store total accuracy in list

```
// Static variables to store total accuracy and list
count
public static double totalAccuracy { get; set; }
public static double listCount { get; set; }
```

Then we use 'AnomalyPlotter' class is used to plot graphs of sequences of data and their anomalies. The class contains one static method, PlotGraphWithAnomalies, which takes two parameters: allData: a list of arrays of doubles, where each array represents a sequence of data, and allAnomalyIndices: a list of lists of integers, where each list represents the indices of the anomalies in a sequence [Listing 16].

Listing 16 Plot anomalies with line graph

```
public static void
PlotGraphWithAnomalies(List<double[]>
allData, List<List<int>> allAnomalyIndices)
```

The method works by creating a line graph for each sequence and a scatter plot for its anomalies. It then combines all the graphs into a chart and displays it. This method uses the XPlot.Plotly library to create the graphs and the chart.
Two lists, allGraphs and allAnomalies, are initialized to store the graphs for the data sequences and their anomalies respectively [Listing 17].

Listing 17 Store graphs and Anomalies in list

```
List<Scatter> allGraphs = new List<Scatter>();
List<Scatter> allAnomalies = new
List<Scatter>();
```

For each sequence in allData and its corresponding anomaly indices in allAnomalyIndices, a graph for the sequence and a graph for the anomalies are created [Listing 18].

Listing 18 Looping through data and anomaly indices

```
for (int i = 0; i < allData.Count; i++)
{
  double[] data = allData[i];
  List<int> anomalyIndices =
allAnomalyIndices[i];
  ...
```

A Scatter object graph is created for the sequence. The x-values are the indices of the data points, and the y-values are the data points themselves. The graph is a line graph and is named "Sequence" followed by the index of the sequence [Listing 19].

Listing 19 Scatter object graph

```
var graph = new Scatter
{
    x = Enumerable.Range(0,
data.Length).ToArray(),
    y = data,
    mode = "lines",
    name = "Sequence" + i
};
```

Another Scatter object anomalies is created for the anomalies in the sequence. The x-values are the indices of the anomalies, and the y-values are the values of the anomalies. The graph is a scatter plot and is named "Anomalies in sequence" followed by the index of the sequence. The color of the markers is set to red [Listing 20].

Listing 20 all scatter object graph

```
var anomalies = new Scatter
{
    x = anomalyIndices.ToArray(),
    y = anomalyIndices.Select(index =>
data[index]).ToArray(),
    mode = "markers",
    name = "Anomalies in sequence" + i,
    marker = new Marker { color = "red" }
};
```

In this section graph and anomalies are added to allGraphs and allAnomalies respectively [Listing 21].

Listing 21 added anomalies and graph

```
allGraphs.Add(graph);
allAnomalies.Add(anomalies);
```

A Chart object chart is created by plotting all the graphs in allGraphs and allAnomalies. The title of the chart and the labels of the x-axis and y-axis are set. Finally, the chart is displayed [Listing 22].

Listing 22 Plot all anomalies graph

```
var chart =
Chart.Plot(allGraphs.Concat(allAnomalies));
chart.WithTitle("Graph with Anomalies");
chart.WithXTitle("X-axis(Index of data point)");
chart.WithYTitle("Y-axis(Value of data point)");
chart.Show();
```

III. RESULTS

Once the experiment concludes, the anomaly detection results are persisted to the screen, along with HTM accuracy for each individual number sequences and overall HTM accuracy for the whole experiment, like this. After experiment is completed, the plotted graph pops up in the default browser like this. Red dots mark the anomalies in our numerical sequence data from predicting folder. A sample plotted graph is given below for a single experiment run (Figure 1).

Overall, we conducted anomaly detection experiments on two distinct datasets.

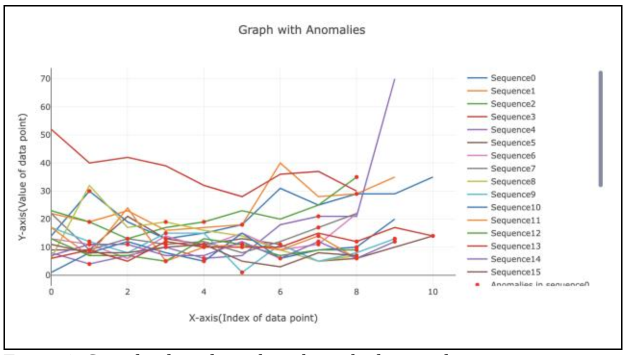

Figure 1: Sample plotted graph with marked anomalies

For a single experiment, we have taken average HTM accuracy for all the tested sequences. For instance, if we are testing three sequences in an anomaly detection experiment, if the average HTM accuracy of a three tested lists are 35%, 25% and 30%, the average HTM accuracy for that experiment should be 30%.

We executed three experiments with varying tolerance values for the NAB dataset (Real data) and two experiments for the Weather Fabricated dataset (Fabricated). We have taken average of all the experiment runs, and our results indicated an average accuracy (for three experiments) of *22.58%* for the NAB dataset and *25.75%* (for two experiments) for the weather dataset.

All the output logs and graphs are stored in our project output folder.

IV. Conclusion

Overall, in conclusion, we can say that the HTM accuracy can be improved by using more data. We were not able to train large amounts of data due to limitations of our local machine. However, we can try using more number of numerical sequences in cloud.

References

[1] J. Hawkins and S. Blakeslee, "On Intelligence", Henry Holt, New York, 2004.

[2] J. Struye and S. Latré, "Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons," Neurocomputing, vol. 396, pp. 291–301, Jul. 2020, doi: https://doi.org/10.1016/j.neucom.2018.09.098.

[3] J. A. Starzyk, and H. He, "Spatio–Temporal Memories for Machine Learning: A Long-Term Memory Organization", IEEE TRANSACTIONS ON NEURAL NETWORKS, vol. 20, no. 5, pp. 68-780, 2009.

[4] J. Wu, W. Zeng, and F. Yan, "Hierarchical Temporal Memory method for time-series-based anomaly detection," Neurocomputing, vol. 273, pp. 535–546, Jan. 2018, doi: https://doi.org/10.1016/j.neucom.2017.08.026.

[5] "A Machine Learning Guide to HTM (Hierarchical Temporal Memory)," Numenta. https://www.numenta.com/blog/2019/10/24/machine-learning-guide-to-htm/, (Last Accessed: 22.03,2024).

[6] "NAB/data/realTweets/Twitter_volume_GOOG.csv at master · numenta/NAB," GitHub. https://github.com/numenta/NAB/blob/master/data/realTweets/Twitter_volume_GOOG.csv (Last Accessed: 24.03,2024).

[7] AccuWeather, "Dhaka, Dhaka, Bangladesh Monthly Weather," Accuweather.com, 2024. https://www.accuweather.com/en/bd/dhaka/28143/march-weather/28143?year=2023 (accessed: 26.03. 2024).

[8] S. Purdy, "Encoders Encoding Data for HTM Systems." Available: https://www.numenta.com/assets/pdf/biological-and-machine-intelligence/BaMI-Encoders.pdf, (Last Accessed: 24.03,2024).