*ML22/23-12: Implement Anomaly Detection Sample*

|  |  |
| --- | --- |
| Md Sohel Rana  sohel.rana@stud.fra-uas.de | Md Ashiqur Rahman  ashiqur.rahman@stud.fra-uas.de |

*Abstract*—HTM (Hierarchical Temporal Memory) is an impactful machine learning algorithm approach that is biologically inspired in both aspects, structurally and functionally, by the neocortex of a human brain that processes time series data in a distributed manner using a hierarchical network of nodes. HTM works in a decentralized manner with the help of a tiered arrangement that allows each node and column to learn and recognize patterns in input data. This feature enables actions such as processing information, recognizing and identifying patterns, and making future predictions based on previous learning. This is a potential approach which can be used for anomaly detection and prediction in numerous sectors such as healthcare, finance, geological disasters, cyber-intrusion detection, military surveillance, system fault detection. This paper presents an anomaly detection sample using an HTM model trained on multiple simple numeric integer sequences. This model learns patterns from the input data and identifies anomalies by comparing real data with predicted data from learning within a set tolerance threshold. The paper also provides a detailed explanation of anomaly detection techniques with algorithm implementation.

Keywords—HTM, anomaly detection, machine learning, multi-sequence learning, NeoCortex API.

# Introduction

Hierarchical Temporal Memory (HTM), a biologically constrained machine intelligence technique, was created by Numenta. It was first published in 2004 by Sandra Blakeslee and Jeff Hawkins, a brain scientist and the founder of the Redwood Neuroscience Research Institute [1]. This machine learning algorithm works based on the theory of how the biological neocortex works, and this approach basically depends on principles of the Thousand Brains Theory. The fundamental of this approach is responsible for higher order processes like language, conscious movement and thought, and sensory perception [2]. HTM design and operation are modeled after the neocortex, a sizable, intricate region of the human brain. HTM aims to replicate the same fundamental neocortical processes by recognizing complex temporal patterns and correlations in data and making future predictions from them [3].

Hierarchy in HTM refers to the layered structure of a neural network, which consists of multiple layers of neurons. Each layer performs a specific type of computation, and information is passed on from lower layers to higher layers for further processing. The lower layers receive input from the environment, such as sensory data, and encode the input into a distributed representation in these layers. It is especially well-suited for sequence learning modeling, similar to RNN methods, like Long Short-Term Memory (LSTM) and Gated Recurrent unit (GRU) [3].

HTM can be viewed as a specific kind of hierarchical Bayesian model. It also uses spatial-temporal theory to learn the structure and invariance of the space of problems [4]. By following its characteristics HTM has also been applied to anomaly detection in recent years. An anomaly is something that deviates from the typical or expected state. Anomalies are sometimes referred to as outliers, discordant observations, exceptions, aberrations, surprises, etc. Finding anomalous patterns in data is known as anomaly detection. A large portion of the data in the world is time-series, streaming data, and in crucial situations, anomalies can provide important information. Examples of this can be found in a variety of industries, including energy, IT, security, finance, and medicine. It's challenging to find anomalies in streaming data, detectors must process data in real-time rather than in batches and learn while making predictions. The effectiveness of real-time anomaly detectors cannot be sufficiently tested or scored using any benchmarks [5]. The ideal detector would function with real-world time-series data across several domains, identify all abnormalities as quickly as feasible, avoid false alarms, and automatically adjust to changing statistics [4]. Because the Hierarchical Temporal Memory Cortical Learning Algorithm (HTM CLA) has most of the properties, its use in anomaly detection is becoming more and more popular.

In HTM CLA several essential elements are included to handle input data. The raw input data is first encoded and transformed into a sparse distributed representation(SDR) using an encoder. This SDR, which includes binary information with few active bits, is made more robust to noise by passing it via a spatial pooler. The Temporal Memory component, which is responsible for recognizing and detecting patterns in the data, then processes the output. The Classifier component uses these learned patterns to classify input data and predict new patterns. Additionally, over time, the system will continuously learn new patterns thanks to the Homeostatic Plasticity Controller [3].

Figure 1 shows how input data is processed in an HTM system.

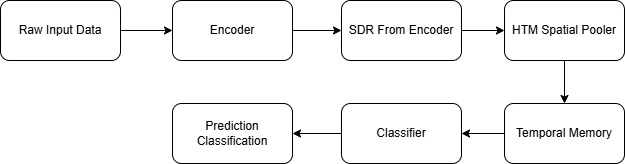


Figure 1: How HTM System works

The temporal memory layer of the HTM network processes the encoded input data following spatial pooling. Capturing and memorizing temporal patterns and sequences in the input data is the responsibility of temporal memory. By creating predictive links between active cells in various time steps, it keeps a predictive model of the input stream [3].

Based on the present input and past context, the HTM network can predict and infer future states after learning temporal patterns in the input data. To predict the next probable input in the sequence, the network uses the input data to activate predictive cells during inference [3].

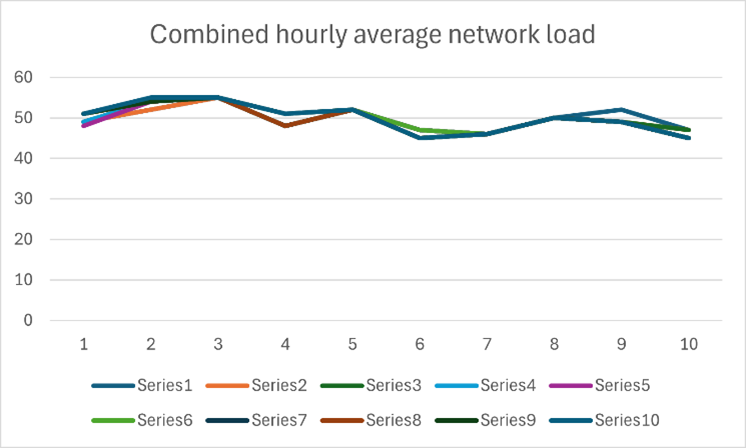
HTM systems use feedback mechanisms to continuously learn and adjust to shifting input patterns. The network strengthens its ties and gains knowledge from accurate predictions when the expected and real inputs match. On the other hand, the network adjusts its connections to enhance subsequent forecasts in the event of a prediction error [3].

HTM systems can detect anomalies or departures from expected input patterns in addition to inference and prediction. When the input data deviates substantially from the taught prediction model, it is considered an anomaly and may indicate uncommon or unexpected events [3].

# Methodology

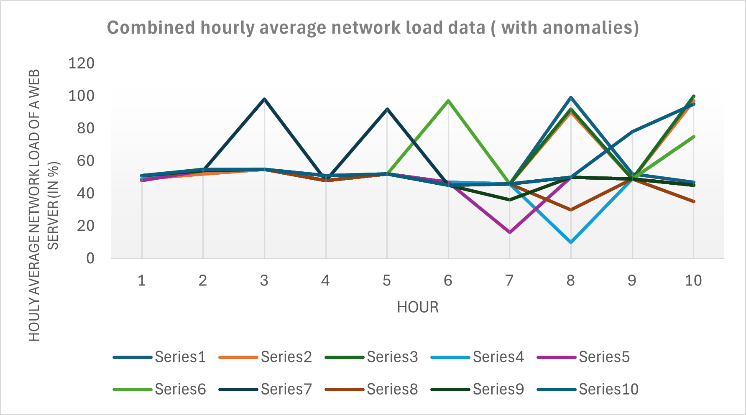
To detect Anomalies in our project we used the Neocortex API [1], which was developed within the .NET framework, we also used Hierarchical Temporal Memory (HTM) for its unique functionality [3]. For training and testing our project, we are going to use artificially generated data, which contains numerous samples of simple integer sequences in the form of (1,2,3,..). These sequences will be placed in a few commas separated value (CSV) files. There will be two folders inside our main project folder, training and predicting. These folders will contain a few of these CSV files. The predicting folder contains data like training (shown in figure 2), but with added anomalies randomly added inside it. We are going to read data from both the folders and train our HTM model using it. After that we are going to take a part of numerical sequence, trim it in the beginning, from all the numeric sequences of the predicting data and use it to predict anomalies in our data which we have placed earlier, and this will be automatically done, without user interaction.

We are using artificially generated network traffic load data (in percentage, rounded to the nearest integer) from a sample web server. The values are taken over time form numerical sequence.



**Figure 2: Graph of numerical sequences without anomalies which will be used for our training HTM model.**

For our test we will consider values between [45,55] as normal , and anything outside this range as anomalies . The predicting folder contains data with random anomalies placed at different positions, with values between [0,100]. The combined data from both the training folder and predicting folder is shown in Figure 3.



**Figure 3: Graph of all numerical sequences with anomalies.**

We will use the **MultiSequenceLearning** class from the NeoCortex API as the foundation of our project. It will help us train the HTM model and make predictions. The class works like this:

a. First, HTM configuration is set, and memory connections are initialized. Then, the HTM Classifier, Cortex Layer, and Homeostatic Plasticity Controller are set up.

b. Next, the Spatial Pooler and Temporal Memory are initialized.

c. The spatial pooler memory is added to the cortex layer and trained for the maximum number of cycles.

d. Then, temporal memory is added to the cortex layer to learn all the input sequences.

e. Finally, the trained cortex layer and HTM classifier are returned.

We need to pass Encoder and HTM Configuration settings to the relevant components in this class. We will use the classifier object from the trained HTM model to predict values, which will then be used for anomaly detection.

We will train and test data with integer values ranging from 0 to 100, without periodicity. The configuration settings are shown in Listing 1. We will use 21 active bits for representation. There are 101 values representing integers between 0 and 100. The total input bits are calculated as:

n = buckets + w – 1 = 101 + 21 - 1 = 121.

int inputBits = 121;

int numColumns = 1210;

------------------------------------------------------------------

double max = 100;

Dictionary<string, object> settings = new Dictionary<string, object>()

{

{ "W", 21},

{ "N", inputBits},

{ "Radius", -1.0},

{ "MinVal", 0.0},

{ "Periodic", false},

{ "Name", "integer"},

{ "ClipInput", false},

{ "MaxVal", max}

};

Listing 1: Encoder settings for our project

The minimum and maximum values are set at 0 and 100, as all expected values fall within this range. If the input data has a different range, these values should be adjusted accordingly. We have kept the default HTM configuration unchanged.

Our project follows these steps:

a. The ReadFolder method from the CSVFolderReader class reads all files in a folder. Alternatively, the ReadFile method from CSVFileReader reads a single file. Both store the data as a list of numeric sequences for later use.These classes include exception handling to manage non-numeric data. The TrimSequences method is used in our unsupervised approach. It randomly removes 1 to 4 elements from the start of a numeric sequence and returns the trimmed version. Both methods are shown in Listing 2.

public List<List<double>> ReadFolder()

{

-------

return folderSequences;

}

public static List<List<double>>

TrimSequences(List<List<double>> sequences)

{

-----

return trimmedSequences;

}

Listing 2: Important methods in CSVFolderReader class

b. Next, the BuildHTMInput method from the CSVToHTM class converts the read sequences into a format suitable for HTM training. This is shown in Listing 3.

public Dictionary<string, List<double>> BuildHTMInput(List<List<double>> sequences)

{

Dictionary<string, List<double>> dictionary = new Dictionary<string, List<double>>();

for (int i = 0; i < sequences.Count; i++)

{

// Unique key created and added to dictionary for HTM Input

string key = "S" + (i + 1);

List<double> value = sequences[i];

dictionary.Add(key, value);

}

return dictionary;

}

Listing 3: BuildHTMInput method

c. Then, the RunHTMTraining method from the HTMTraining class trains the model using the MultiSequenceLearning class, as shown in Listing 4. It combines numerical sequences from both the training and predicting folders to train the HTM model. This class returns the trained model object, predictor, which will later be used for prediction and anomaly detection.

.....

MultiSequenceLearning learningAlgorithm = new MultiSequenceLearning();

trainedPredictor = learningAlgorithm.Run(htmInput);

.....

Listing 4: Code demonstrating how data is passed to HTM model using instance of class multisequence learning.

# Results

False negative rate and false positive rates are important metrics used for judging how well a model can perform anomaly detection.

**False Negative rate, or, FNR = FN / (FN + TP)**

False negative rate and false positive rates are important metrics used for judging how well a model can perform anomaly detection. where FN represents the number of false negatives, or true anomalies that are mistakenly classified as normal, and TP represents the number of true positives, or true anomalies that are correctly classified as anomalies.

**False Positive rate, or, FPR = FP / (FP + TN)**

where TN is the number of true negatives, or the number of normal observations that are correctly classified as normal, and FP is the number of false positives, or the number of normal observations that are mistakenly identified as anomalies.

Let us discuss the output of this experiment. For a brief analysis, we are going to discuss a part of our output text. If the sequence passed to our trained HTM engine is [54, 55, 48, 52, 47, 16, 50, 49, 45], we get the following output with respective accuracies.

Start of the raw output:

Testing the sequence for anomaly detection: 54, 55, 48, 52, 47, 16, 50, 49, 45.

First element in the testing sequence from input list: 54

No anomaly detected in the first element. HTM Engine found similarity to be:62,79%. Starting check from beginning of the list.

Current element in the testing sequence from input list: 54

Anomaly not detected in the next element!! HTM Engine found similarity to be: 42,86%.

Current element in the testing sequence from input list: 55

Anomaly not detected in the next element!! HTM Engine found similarity to be: 92,59%.

Current element in the testing sequence from input list: 48

Anomaly not detected in the next element!! HTM Engine found similarity to be: 100%.

Current element in the testing sequence from input list: 52

\*\*\*\*Anomaly detected\*\*\*\* in the next element. HTM Engine predicted it to be 97 with similarity: 100%, but the actual value is 47.

As anomaly was detected, so we are skipping to the next element in our testing sequence.

Current element in the testing sequence from input list: 16

Anomaly not detected in the next element!! HTM Engine found similarity to be: 100%.

Current element in the testing sequence from input list: 50

Nothing predicted from HTM Engine. Anomaly cannot be detected.

Current element in the testing sequence from input list: 49

\*\*\*\*Anomaly detected\*\*\*\* in the next element. HTM Engine predicted it to be 75 with similarity: 55,81%, but the actual value is 45.

As anomaly was detected, so we are skipping to the next element in our testing sequence.

After running our sample project, we analyzed the output and got the following results:

* Average FNR of the experiment: **0**
* Average FPR of the experiment: **0**

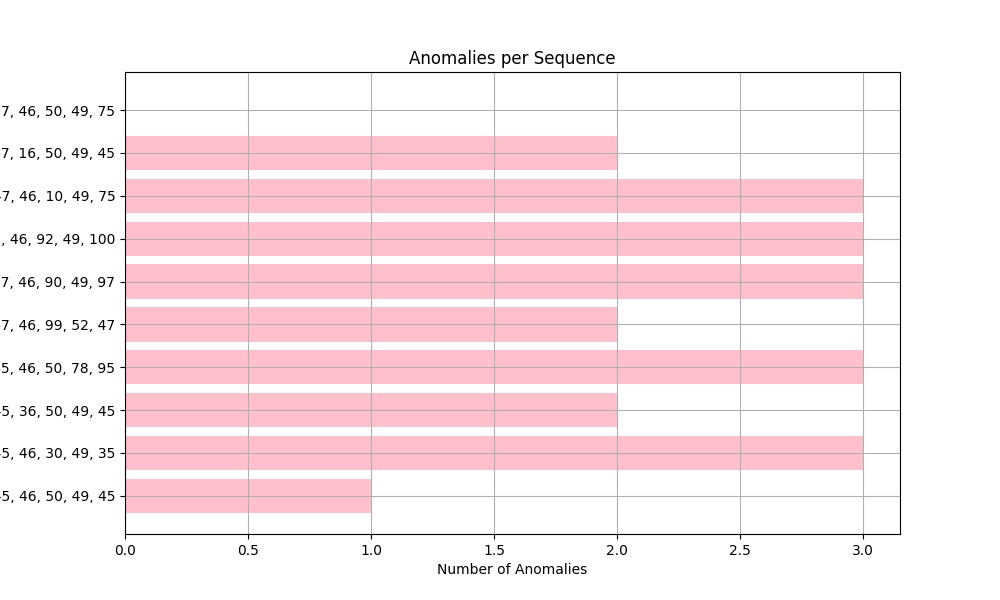


Figure 2: Anomalies per sequence

It visualizes the number of anomalies detected in different sequences of numerical data. The x-axis represents the number of anomalies, while the y-axis lists different sequences of numerical values. The number of anomalies varies across different sequences, with some sequences showing higher anomaly counts than others. Each sequence consists of five numerical values, suggesting that the dataset comprises multiple five-element sequences where anomaly detection was performed. The horizontal alignment of the bars provides a clear comparison between sequences, showing which sequences contain more anomalies. The variation in anomaly counts suggests heterogeneous patterns within the sequences. Some sequences show higher anomaly counts, which may indicate recurring systematic errors, outliers, or deviations in data behavior. The sequences with fewer anomalies might represent normal patterns or stable trends in the dataset.

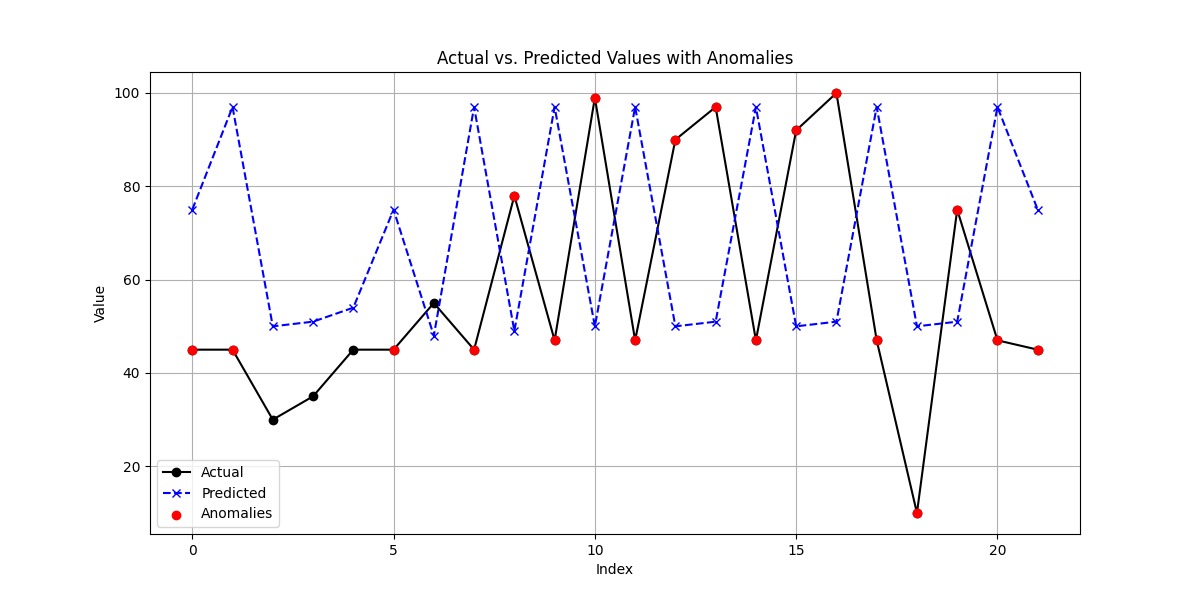


Figure 3: Actual vs. predicted values with anomalies

It is a time-series comparison plot showing the relationship between actual values, predicted values, and detected anomalies over a sequence of indexed data points. The actual values, represented by a black solid line, show fluctuations over time, while the predicted values, depicted as a blue dashed line, indicate the expected trend. Significant deviations between these two lines are marked as anomalies with red dots, suggesting instances where the actual values diverge notably from predictions. These anomalies may result from unexpected real-world events, sensor malfunctions, or model inaccuracies. The pattern of detected anomalies suggests that the predictive model struggles to capture sudden spikes or drops, indicating potential limitations in forecasting accuracy.

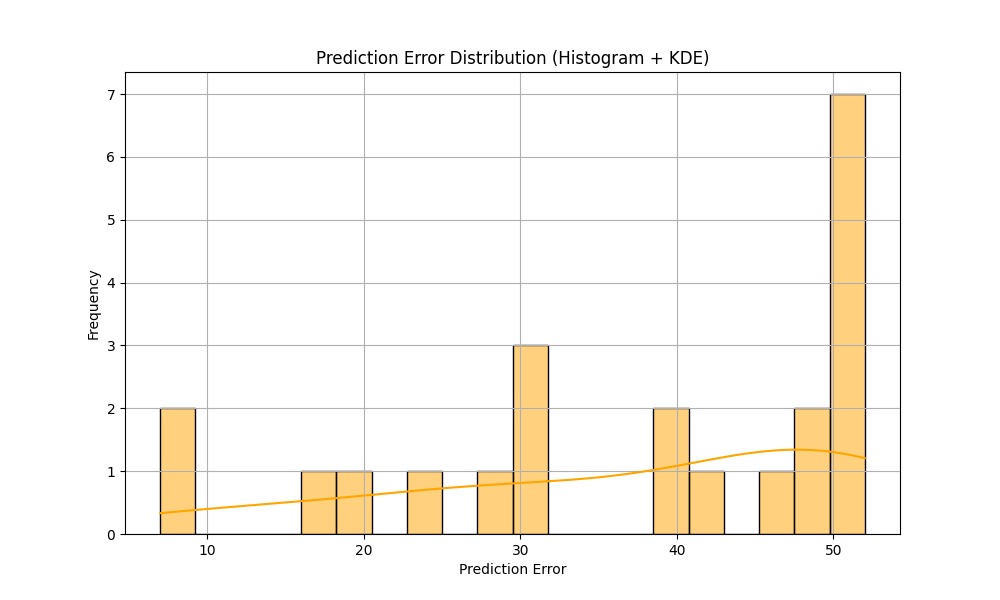


Figure 4: Prediction error distribution(Histogram + KDE)

It represents the distribution of prediction errors using a histogram overlaid with a Kernel Density Estimate (KDE) curve, providing insight into the error distribution pattern. The x-axis denotes the prediction error, while the y-axis represents its frequency. The histogram bars, shaded in light orange, show the count of occurrences for different error ranges, while the KDE curve smooths the distribution to reveal underlying trends. The presence of multiple peaks and a higher frequency of large errors suggests that the model exhibits varying degrees of inaccuracies, with some instances experiencing significantly large prediction errors. This skewed distribution may indicate the presence of systematic biases, model inefficiencies, or data inconsistencies, which should be further investigated to enhance predictive accuracy.

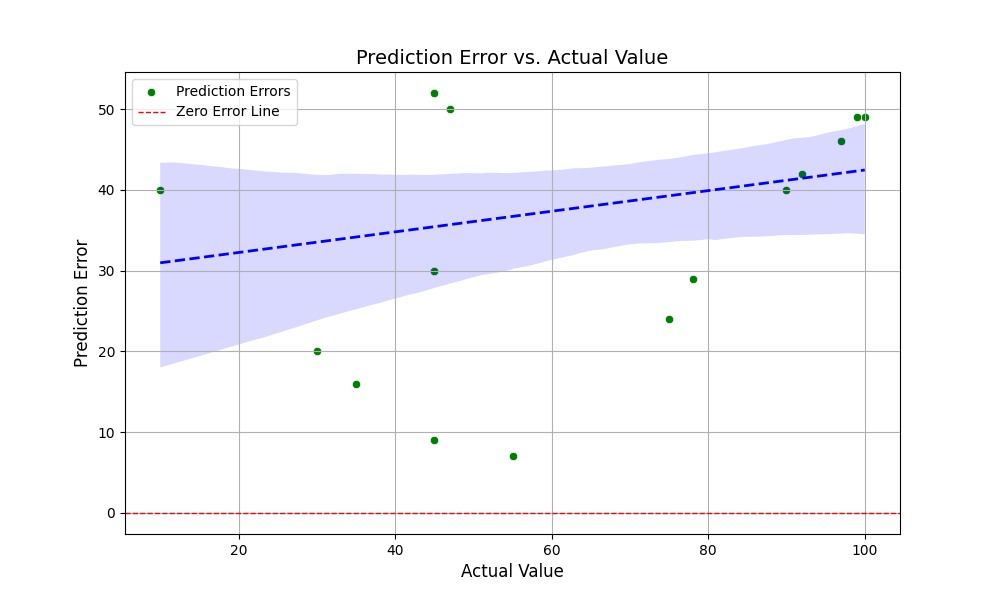


Figure 5: Prediction error vs. actual value

It represents the relationship between prediction error and actual values in a predictive model. The x-axis denotes the actual values, while the y-axis indicates the prediction error. Green dots represent the individual prediction errors, showing how far off the model’s predictions are from actual values. A blue dashed regression line with a shaded confidence interval suggests a positive correlation, meaning that as actual values increase, prediction errors tend to increase as well. The red dashed line at y = 0 represents an ideal scenario where prediction errors are zero. The dispersion of green points above the red line suggests the model consistently overestimates actual values. The increasing trend in errors highlights potential bias or inaccuracy in the model, which may require further optimization to improve predictive performance.

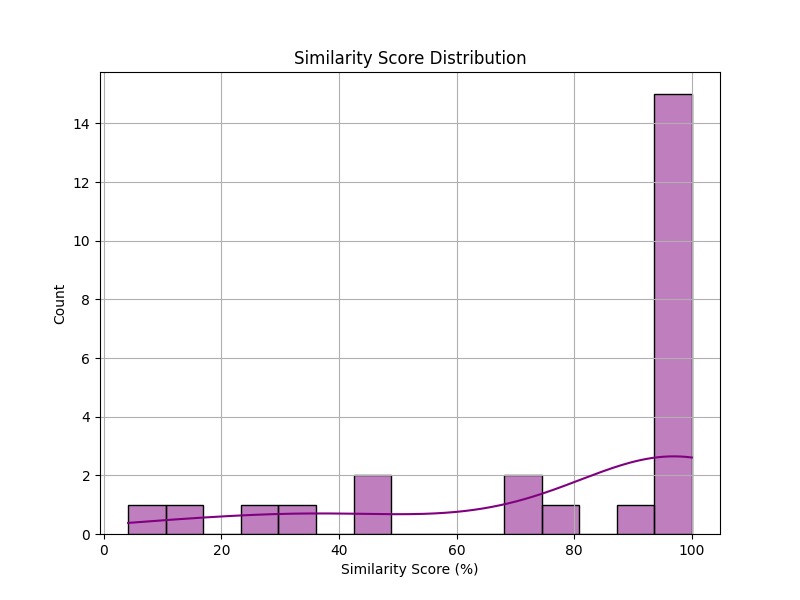


Figure 6: Similarity score distribution

It illustrates the distribution of similarity scores in percentage, with the x-axis representing similarity scores and the y-axis showing the count of occurrences. The histogram, colored in purple, indicates that most of the similarity scores are concentrated near 100%, suggesting a high frequency of near-identical matches. A few lower similarity scores are scattered across the lower percentage ranges, but they appear less frequently. The smoothed density line overlaid on the histogram suggests a rising trend towards higher similarity scores. This pattern implies that a majority of the compared data points exhibit strong similarity, while only a small portion shows low similarity. Such a distribution could indicate that the dataset contains a significant number of highly similar or duplicate entries.

# Discussion

##### References

|  |  |
| --- | --- |
| [1] | J. H. a. S. Blakeslee, "On Intelligence," in *Henry Holt*, New York, 2004. |
| [2] | J. S. a. S. Latré, "Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons," in *Neurocomputing*, 2020. |
| [3] | V. Lomonaco, "A machine learning guide to HTM (Hierarchical Temporal Memory)," 2019. [Online]. Available: https://www.numenta.com/blog/2019/10/24/machine-learning-guide-to-htm. |
| [4] | a. H. H. J. A. Starzyk, "Spatio–Temporal Memories for Machine Learning: A Long-Term Memory Organization," in *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 2009. |
| [5] | A. &. A. S. Lavin, "Evaluating real-time anomaly detection algorithms - the Numenta Anomaly Benchmark," in *In arXiv [cs.AI]*, 2015. |