Investigate label prediction from the time-series sequence

Md Rizwanul Islam
md.islam9@stud.fra-uas.de

Md Rabiul Islam
md.islam3@stud.fra-uas.de

Md Jubayar Hosan
md.hosan@stud.fra-uas.de

Abstract— Label prediction from time-series sequence has been widely applied to the finance industry in applications such as stock market price and commodity price forecasting. In recent years, machine learning algorithms have become increasingly popular in label prediction from time-series sequences. There is a time-series sequence of values in many industrial scenarios, for example, a sequence that shows the number of taxi pick-ups at a specific time. A popular problem is how to label time series data to measure the forecast accuracy of machine learning models and, as a result, ultimate investment returns. Existing time series labeling approaches mostly label data by comparing current data to data from a short period in the future. This paper represents an investigation of label prediction from the time series sequence and implements the code for multisequence learning and prediction to address the above problem. The sequence in HTM is learned by the htmClassifier.Learn() method. For this approach it assumes that the label can be the number of passengers and the second argument of the Learn function is the encoded time-series value. Then it encodes the date and time as a combined encoding of two scalar values and reduces the time precision to a specific time using segmentation of one hour. After that, it predicts using the trained model, then examines the outcome and calculates the accuracy. The results of the paper also proved that the label prediction from the time series data set using multisequence learning is the best choice for dealing with the prediction of a specific time.

Keywords— HTM CLA, HTMConfig, Spatial Pooler (SP), Temporal Memory (TM), HTM Classifier, Sparse Distributed Representation (SDR), Cortex Layer, Active Cell, Predictive Cell, Htm Prediction Engine

I. INTRODUCTION

A time-series sequence is a set of observations, each one being recorded at a specific time [1]. Prediction of time series sequence data is a relatively complex task. Since there are many factors affecting time series data, it is difficult to predict the trend of time series data accurately. Investigate label prediction from a time-series sequence aims at solving various problems, specifically in industrial scenarios because there is a time-series sequence of values in many industrial scenarios. Many studies have shown that the market is non-linear and chaotic, especially for the time-series data such as the values of stocks, foreign exchanges, and commodities in the financial market that are sensitive to external impact and tend to fluctuate violently [2]. Such time series data often have strong non-linear characteristics. How to better predict the trend of the financial market is of great significance for reducing investment risk and making financial decisions. Time series data, on the other hand, are frequently non-linear, with clear short-term unpredictability. As a result, the continuous trend aspects of the time series sequence of data were not captured by these labeling approaches, resulting in a discrepancy between their labeling findings and the actual data [3]. There is a time-series sequence of values in many industrial contexts, such as a sequence that indicates the number of taxi pick-ups at a certain moment. How to label time-series data to quantify the accuracy of machine learning models and, as a result, final investment returns is a common challenge [4]. Existing time series labeling methods primarily compare present data to data from a short time in the future to classify data. This study investigates label prediction from a time series sequence and implements the code for sequence learning and prediction to solve the aforesaid challenge. For this approach, primarily started working with a CSV file containing sample data (Taxi-TLC Trip Record Data) that is mostly unsorted. After that read the passenger data from the unsorted sample data set and prepare a new CSV file. It is known to everyone that every sample data set has many columns. However, for this paper, the lpep_pickup_datetime and passenger_count columns are important to consider from the sample data set. The pickup date-time of passengers is then segmented for each 1 hour and recorded in a processed CSV file because every 60 minutes, one segment of a given date is considered. Then read data from the modified CSV file and encode it. For encoding, these four encoders dayEncoder, monthEncoder, segmentEncoder, and dayofWeek have been used to train data. Please note that because the year is static, it is not taken into consideration during encoding. After that htmClassifier learns the sequence in HTM. The sequence in HTM is learned (simplified) by using the following statement:

```
htmClassifier.Learn(label, sdr of ([t, day of week]))
```

Listing 1: HTM Learn Method from project overview docx (Project No. 24)

The function Learn () is used to learn something new. The label can be the number of passengers in this manner, and the encoded time-series value is the second parameter of the Learn function. The date and time are then encoded as a combination of two scalar values, and the time precision is

reduced to a precise time via segmentation. The learned model is then used to make predictions, after which the outcome is analyzed, and the prediction accuracy is calculated. The user can input any date, time, or date-time segment, and htmClassifier will return a predicted result. The second task is then used the learned model for prediction. The user enters the time segment (01-01-2022 00:18) and the classifier is used to predict the number of passengers by using the following statement:

```
var predictedValues =
htmClassifier.GetPredictedInputValues(encoded time
segment).
```

Listing 2: Prediction code

GetPredictedInputValues is a function that returns the predicted input values for an encoded time-segment A date-time of 000date0000time00 is considered. In this paper, the main objective is to see if label prediction from a time series data set using multisequence learning is the best option for dealing with time prediction.

II. Methodology

The main motivation behind label prediction from time-series sequence is to let the machine perform progressed functionalities [5] on multisequence learning and prediction. So that it can learn the sequence and the learned model is then used to make predictions. Apart from that, there are currently several studies looking into investigating label prediction from time-series sequences on which some people have previously worked and others on which work is still ongoing to deal with the prediction of a specific time with NeocortexApi. So, investigating label prediction from time-series sequence is intended for this experiment.

This section is divided into three sub-sections. The proposed architecture for the investigation process of the label prediction from time-series sequences be described in detail in the first sub-section, then the following sub-section will describe the whole algorithmic process in pseudo-code to implement the proposed label prediction from time-series sequences architecture and the final sub-section will explain the creating process of label prediction from time-series sequences in C# code from the proposed algorithm by using NeocortexApi v.1.0.15 and Microsoft .Net 6.0

A. *Proposed Architecture for label prediction from the time-series sequence*

HTM can be described as the theory that attempts to describe the functioning of the neocortex, as well as the methodology that intends to provide machines with the capacity to learn in a human way.

Time series sequence using HTM is to let the machine perform progressed functionalities and cognitive tasks the same as the human brain. This time-based prediction system is planned to perform future information expectations based on learning and memorizing the information already fed into it. For HTM to memorize and anticipate, it needs input from the user. The input can be within the form of a scalar value, date-time string, or a picture. The encoder is at that point utilized to change over the input into SDR which can be learned by the HTM classifier. Depending on the encoder, the value can be anything i.e., scalar value, time, day in a week, geo-location, etc. The SDR is regularly a cluster in '1' and '0'. The Spatial Pooler is an algorithm inside of HTM that's able to learn spatial patterns. The Spatial Pooler regularly gets a cluster of bits as input and changes over it into the SDR. The input of the Spatial Pooler is as a rule the array changed over by some encoder, or bits of an image. Another Portion of HTM CLA is Temporal Memory. The input of Temporal Memory is the SDR output produced by Spatial Pooler. Temporal memory is an algorithm that learns arrangements of Sparse Distributed Representations (SDRs) shaped by the Spatial Pooling algorithm and makes expectations of what another input SDR will be.

At each discrete time instance, HTM carries out three steps on the new input. The following descriptions of the steps, detailed in the next sections, are taken from:
Step 1: Create an SDR of the input by activating whole columns.
Step 2: Place the input in context by selecting among cells in an active column.
Step 3: Predict future patterns from learned transitions between SDRs.

Spatial pooler creates SDR of an input, during which active columns of cells are mapped. Each column has connections with a subregion of input bits via synapses on the proximal dendrite. Several columns may be identical, but they are not the same. Different patterns generate different levels of activation of the columns. The columns with the strongest activation restrain columns with lower activation. The area of interference area around a column is adjustable and can span from very small to the entire region. The inhibitory mechanism ensures that the input is represented sparsely. Identical patterns generated should generate identical activated columns. HTM learns by training and unforming connections between cells. Learning occurs when the synapses' permanence values are updated. Only the active columns increase the persistence value of synapses associated with active bits, whilst the other columns decrease it. Columns that are not active for an extended length of time do not learn anything. To avoid wasting columns, their overlap scores are "boosted" to ensure that all columns participate in pattern learning.

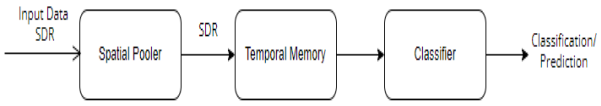


Figure 1: HTM Algorithm Flow.

For the following experiment, all date-time are elements of a single sequence. Here HTM learns each element of sequence i.e., date with its label is passenger count in our case. We have trained SP alone on all inputs till HPA has seen all patterns and achieved a stable state. Then we have added TM to the cortex layer and trained the two components together. When HTM is trained on the sequence, it can then predict the next element i.e., the next date passenger count.

B. Implementation Algorithm based on the proposed architecture model in section A :

The algorithm for Multi-sequence Learning is below:

01. Get HTM Config and initialize memory of Connections
02. Initialize HTM Classifier and Cortex Layer
03. Initialize Homeostatic Plasticity Controller
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 segments for multi-sequences using Spatial Pooler
05.02 Continue for the 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 Label 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 the above steps for sequences of multi-sequences for maximum cycles
07. Get the trained Cortex Layer and HTM Classifier

Table 1: Algorithm for multi-sequence learning, See here

Algorithm: Investigate label prediction from the time-series sequence- pseudocode

```
Function RunPassangerTimeSeriesSequenceExperiment()
    List<ProcessedData> ← CSVPRocessingMethods.ProcessExistingDatafromCSVfile(path)

    //Read the taxi data set and write into new processed csv with
 required column
  List<Dictionary<ToString, int[]>> trainTaxiData ← DateTimeEncoders.EncodePassengerData(taxiData)
    // Encode the passenger data based on day month segment and day of week

    EncoderBase encoder ← DateTimeEncoders.FetchDateTimeEncoder()
    // Create multi encoder for day, month, segment, and day of week

   var learningExpResult ← RunExperiment(inputBits, maxCycles, numColumns, encoder, trainTaxiData)
   // Get HtmPredictionEngine after run experiment with trained taxi passanger data uisng sartial pooler and Temporal memory

   PredictPassanagerWithUserInput(learningExpResult)
   // Predict no of passanger based on user Input using trained HtmPredictionEngine

end function
```

Table 2: Pseudocode.

C. Algorithm to Code in C# with NeocortexApi:

This project can be divided into two parts. The first part is data processing, and the Second part is model learning. We are writing all steps to complete this project below and showing the implementation of these given steps with code implementation. To implement our project, all steps are translated to C# code utilizing all available features at NeocortexApi v 1.0.15 package and Microsoft .Net 6.0 framework. The whole project code is open-source, and you can see the current source code at this repository.

The entire code has been split into six stages to clarify the entire conversion phase of pseudocode to C# code in detail:

1. Read all taxi passenger data and process all taxi passenger data for further use
2. Create Encoder based on day, month, segment, and weekday
3. Encode all taxi passengers into SDR format using all four created encoders
4. Enter a newborn stage of the learning algorithm using Spatial Pooler
5. Get an HTM prediction engine using a Temporal memory algorithm
6. Take input and throw it in the HTM prediction engine and get a prediction result

1. Read all taxi passenger data and process all taxi passenger data for further use

The dataset is stored in CSV format. First, it is needed to read data and process data by removing unnecessary data from the data set. Listing 1 reads all data from a CSV file which is a list of data rows. We will iterate every row and process data in our required format. We will take the date time and passenger count. Code for reading of every line of data from the CSV file and process. After reformatting the datetime, we found a processed CSV file 2021_Green_Processed CSV where every dates have 24 segments. Then we grouped by the date and segment for making a sequence. We have considered every date as a sequence which have 24 segments. In the example below, we can see 01/02/2021 is sequence with 24 segments. as seen below:

Sequence 1:

```
01/02/2021 00,212
01/02/2021 01,164
01/02/2021 02,470
01/02/2021 03,420
:       :       :
01/02/2021 21,235
01/02/2021 22,475
01/02/2021 23,454
```

Table 3: Example of a sequence

2. Create Encoder based on day, month, segment, and weekday

Since the experiment has tried with the numeric sequences as the input data set to the encoder. So, an instance of the scalar encoder is needed to declare & initialize before that, which can encode a Numeric (floating point) into an array of bits. In NeocortexApi some default crucial parameters are needed to declare before initializing a scalar encoder like:

- W = Denotes the width, which is the number of bits that are set to encode a single value. It must be set to odd to avoid the centering problem
- N = Denotes the number of bits at the output. Must be N >= W
- Max and Min values = Defines input range. The input sequence at the lower layer must be within that range. The minimum and maximum values are termed in NeoCortex Api as "MinVal" & "MaxVal"
- Radius = Defines (W/N). I where I is the range. So, we set value -1, to neglect the effect of range I

```
ScalarEncoder dayEncoder = new ScalarEncoder(new
Dictionary<string, object>()
        {
           { "W", 3},
           { "N", 35},
           { "MinVal", (double)1}, // Min value = (0).
           { "MaxVal", (double)32}, // Max value = (32).
           { "Periodic", true},
           { "Name", "Date"},
           { "ClipInput", true},
        });
```

Listing 3: Declare encoder and define properties. See here

In Listing 3, The defined encoder name is dayEncoder. Like dayEncoder, four encoders were used in this project. These are dayEncoder, monthEncoder, segmentEncoder and dayOfWeekEncoder.

3. Encode all taxi passengers into SDR format using all four created encoders

Encode the processed taxi passenger data using the created encoder which then creates a stream of bits, a formatted data used in HTM. The code is below:

```
var observationLabel = sequence.Passanger_count;
int day = sequence.Date.Day;
int month = sequence.Date.Month;
int segement = Convert.ToInt32(sequence.Segment);
int dayOfWeek = (int)sequence.Date.DayOfWeek;

int[] sdr = new int[0];

sdr = sdr.Concat(dayEncoder.Encode(day)).ToArray();
sdr = sdr.Concat(monthEncoder.Encode(month)).ToArray();
sdr =
sdr.Concat(segmentEncoder.Encode(segement)).ToArray();
sdr =
sdr.Concat(dayOfWeekEncoder.Encode(dayOfWeek)).ToAr
ray();
```

Listing 4: Convert input data into SDR format using Decoder. See here

4. Enter a newborn stage of the learning algorithm using Spatial Pooler

In this stage, we do not add here TM in the layer. This is omitted for practical reasons because we first enter the newborn-stage of the algorithm. In this stage we want that SP get boosted and see all elements before we start learning with TM. All would also work fine with TM in layer, but it would work much slower. So, to improve the speed of experiment, we first omit the TM and then after the newborn-stage we add it to the layer. SP gets boosted and sees all elements before it started learning with TM. Use maxCycles to decide how many times it iterates to get stable. In Listing 5, it got SDR to use in the TM algorithm to learn the machine.

```
var lyrOut = layer1.Compute(elementSDR, true);
```

Listing 5: Training SP to get a stable newborn cycle. See here

5. Get an HTM prediction engine using a Temporal memory algorithm

In this stage, we will get an HTM prediction engine using the Temporal Memory algorithm. In listing 6, we will create an object of Temporal Memory and activate it in the cortex layer.

```
var htmConfig = new HtmConfig(new int[] { inputBits },
new int[] { numColumns });
var mem = new Connections(htmConfig);
TemporalMemory tm = new TemporalMemory();
tm.Init(mem);
layer1.HtmModules.Add("tm", tm);
```

Listing 6: Activate the Temporal Memory algorithm. See here

6. Taking the SDR of TM and learn with HTM Classifier

In this stage, we have the SDR value which has been processed by the TM algorithm. Now we have sent the passenger count as observation level and active cells to the HTM classifier for learning the sequence.

```
lyrOut = layer1.Compute(Elements.Value, learn);
cls.Learn(observationLabel, actCells.ToArray());
```

Listing 7: Learning with HTM Classifier here

7. Take input and throw it in the HTM prediction engine and get a prediction result

In listing 8, After completion of learning user can be able to get predicted passenger number of a specified date. User input can be like, dd/MM/yyyy hh:mm. After getting the user input, it encodes in SDR format and throw it into the learning experience machine to get result from learning sequences. If the predicted engine get any matching sequences then it provides the similarity and passenger count.

```
var predictedValuesForUserInput =
learningExpResult.Predict(sdr);
```

Listing 8: Predict results from the learning experience. See here

III. Experiment Results

Accuracy results:

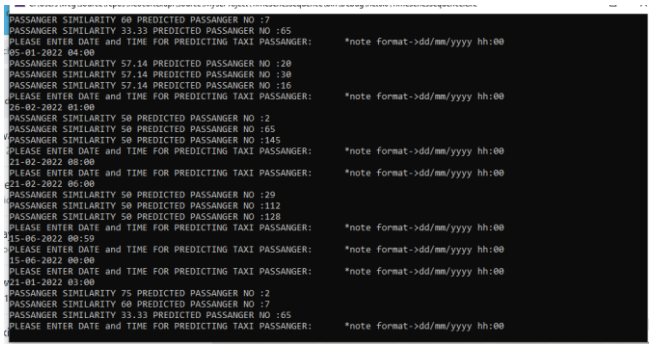
After Processing the data from 2021_Green_Processed CSV and creating the sequences from we have found 216 sequences where every date is a sequence, and every date has 24 segments. As working with 216 sequences are very time consuming, so we consider 50 sequences with 50 cycles for learning and predicting the user input. After learning the sequences, we have tracked the accuracy in a log file. The Output Accuracy Logs for maxCycles50. We are showing here 1 sequence accuracy with 50 cycles:

Sequence Starting for Cycle 1
cycle : 0 Accuracy :4.166666666666666
cycle : 1 Accuracy :45.83333333333333
Cycle: 2 Saturated Accuracy : 45.83333333333333 Number of times repeated 1
cycle : 3 Accuracy :58.333333333333336
cycle : 4 Accuracy :54.166666666666664
cycle : 5 Accuracy :62.5
cycle : 6 Accuracy :70.83333333333334
Cycle: 7 Saturated Accuracy : 70.83333333333334 Number of times repeated 1
cycle : 8 Accuracy :79.16666666666666
cycle : 9 Accuracy :75
cycle : 10 Accuracy :70.83333333333334
cycle : 11 Accuracy :75
cycle : 12 Accuracy :83.33333333333334
Cycle: 13 Saturated Accuracy : 83.33333333333334 Number of times repeated 1
Cycle: 14 Saturated Accuracy : 83.33333333333334 Number of times repeated 2
cycle : 15 Accuracy :75
cycle : 16 Accuracy :87.5
Cycle: 17 Saturated Accuracy : 87.5 Number of times repeated 1
Cycle: 18 Saturated Accuracy : 87.5 Number of times repeated 2
cycle : 19 Accuracy :91.66666666666666
Cycle: 20 Saturated Accuracy : 91.66666666666666 Number of times repeated 1
Cycle: 21 Saturated Accuracy : 91.66666666666666 Number of times repeated 2
Cycle: 22 Saturated Accuracy : 91.66666666666666 Number of times repeated 3
cycle : 23 Accuracy :87.5
cycle : 24 Accuracy :91.66666666666666
Cycle: 25 Saturated Accuracy : 91.66666666666666 Number of times repeated 1
Cycle: 26 Saturated Accuracy : 91.66666666666666 Number of times repeated 2
Cycle: 27 Saturated Accuracy : 91.66666666666666 Number of times repeated 3
Cycle: 28 Saturated Accuracy : 91.66666666666666 Number of times repeated 4
Cycle: 29 Saturated Accuracy : 91.66666666666666 Number of times repeated 5
Cycle: 30 Saturated Accuracy : 91.66666666666666 Number of times repeated 6
Cycle: 31 Saturated Accuracy : 91.66666666666666 Number of times repeated 7
Cycle: 32 Saturated Accuracy : 91.66666666666666 Number of times repeated 8
Cycle: 33 Saturated Accuracy : 91.66666666666666 Number of times repeated 9
cycle : 34 Accuracy :95.83333333333334
cycle : 35 Accuracy :91.66666666666666
Cycle: 36 Saturated Accuracy : 91.66666666666666 Number of times repeated 1
cycle : 37 Accuracy :95.83333333333334
Cycle: 38 Saturated Accuracy : 95.83333333333334 Number of times repeated 1
Cycle: 39 Saturated Accuracy : 95.83333333333334 Number of times repeated 2
Cycle: 40 Saturated Accuracy : 95.83333333333334 Number of times repeated 3
Cycle: 41 Saturated Accuracy : 95.83333333333334 Number of times repeated 4
Cycle: 42 Saturated Accuracy : 95.83333333333334 Number of times repeated 5
cycle : 43 Accuracy :91.66666666666666
Cycle: 44 Saturated Accuracy : 91.66666666666666 Number of times repeated 1
Cycle: 45 Saturated Accuracy : 91.66666666666666 Number of times repeated 2
Cycle: 46 Saturated Accuracy : 91.66666666666666 Number of times repeated 3
Cycle: 47 Saturated Accuracy : 91.66666666666666 Number of times repeated 4
Cycle: 48 Saturated Accuracy : 91.66666666666666 Number of times repeated 5
Cycle: 49 Saturated Accuracy : 91.66666666666666 Number of times repeated 6
Cycle 1 end

For testing purpose, lots of test runs by changing maxcycle and taking different number of sequences. All the logs can be found here.

User input results:

After giving the user inputs as dd/MM/yyyy hh:mm format, user can see the predicated results. Results provide the best 3 results with similarity.



Based on the above result, we can see that user has input following dates from table:

User Date	Predicated passengers	Similarity	Actual passenger
05-01-2022 04:00	20, 30, 16	57.14%, 57.14%, 57.14%	23, 30, 13
26-02-2022 01:00	2, 65, 145	50%, 50%, 50%	70, 190, 6
21-01-2022 03:00	2, 7, 65	75%, 60%, 33.33%	5, 12, 75
21-02-2022 06:00	29, 112, 128	50%, 50%, 50%	32, 120, 150

Table 4: User Input Results

A. Findings

HTM Classifier Limitation

Though some time 90% accuracy has been achieved for experiments but in some experiments this accuracy was not good enough. As in local machine, it is taking lots of time for learning the sequences, so less sequences providing less match and giving low accuracy sometimes. Also noticed accuracy can be differ for the parameters of HTM Config. It returns the 1st matching context instead of looking for the entire array for possible inputs. That's why mismatch happens, and the learning sequence doesn't reach 100% accuracy.

IV. Discussion

In this paper, time series and sequences are thoroughly studied from two aspects, sequence learning, and prediction making. A detailed review summarizing existing approaches is provided. We observe that time series have been studied for decades and various models/learning methods are available but adapting the existing methods to high-dimensional and noisy scenarios remains a challenge [6]. The study on sequences/point processes is relatively new and improvements have been made in recent years. We used HTM sequence memory, a newly created neural network model, to solve real-time sequence learning issues with time-varying input streams in this research. The sequence memory concept is based on cortical pyramidal neurons' computational principles. On real-world datasets, we discussed model performance. The model meets a set of requirements for online sequence learning from input streams with constantly changing statistics, an issue that the cortex must deal with in natural settings. These characteristics determine an algorithm's overall flexibility and its ability to be employed automatically. Even though HTM is still in its infancy compared to other classic neural network models, it satisfies these features and shows promise on real-time sequence learning issues. In the experiments, HTM gives better performance for single element sequences but for sequences that are constituted of multiple elements HTM gives a comparable performance which provides an insight that still there is scope for improvement in that direction. We have used HTM configuration for this experiment which proves there is no need for hyper-parameter tuning for different kinds of data which in this case is not true. Time taken for HTM training is less, as HTM takes the whole sequence set for training which is not time and resource-consuming

A. Future Scopes

Our experiment results show that the HTM algorithm has great potential but still, it is in the infancy stage. These experiments can further be extended for including HTM network state saving checkpoint mechanism, WinForms application for the experiments, Multiple sequence mapper for getting possible class prediction within API, SP+TM state visualization, high order sequence predictions. Advance comparison experiments can be implemented which can compare HTM to others on different kinds of datasets both artificial and real, comparing computing resources, training error, training loss, testing loss, classification/prediction accuracy which can give insights on the limitation of HTM. In cloud server there can be run the experiments with lots of sequences which will produce better match and accuracy. In this project, every date with 24 segments as a sequence has been considered. In future there is a plan to consider every week as a sequence which will consist of 7x24 segments. Thus, lots of segments in one sequence will provide better accuracy and match.

V. References

[1] Li J., Shang P., Zhang X. Financial Time Series Analysis Based on Fractional and Multiscale Permutation Entropy. *Commun. Nonlinear Sci. Numer. Simul.* 2019;78:104880–104892. doi: 10.1016/j.cnsns.2019.104880.

[2] Sapankevych N.I., Sankar R. Time Series Prediction Using Support Vector Machines: A Survey. *IEEE Comput.Intell.Mag.* 2009;4:24–38. doi: 10.1109/MCI.2009.932254

[3] Göçken M., Özçalıcı M., Boru A., Dosdoğru A.T. Integrating Metaheuristics and Artificial Neural Networks for Improved Stock Price Prediction. *Expert Syst.Appl.* 2016;44:320–331. doi: 10.1016/j.eswa.2015.09.029

[4] Continuous online sequence learning with an unsupervised neural network model. Author: Yuwei Cui, Subutai Ahmad, Jeff Hawkins| Numenta Inc.

[5] On the performance of HTM predictions of Medical Streams in real time. Author: Noha O. El-Ganainy, Ilangkp Balasingham, Per Steinar Halvorsen, Leiv Arne Rosseland.

[6] Sequence memory for prediction, inference and behaviour Author: Jeff Hawkins, Dileep George, Jamie Niemasik | Numenta Inc.