Machine Learning Driven Load-Shedding in Parallel Complex Event Processing

Submitted by

Qurat ul Ain Aftab
i17-1001

Supervised by

Dr. Adnan Tariq
Masters of Science (Computer Science)

A thesis submitted in partial fulfillment of the requirements for the degree of
Masters of Science (Computer Science)
at National University of Computer & Emerging Sciences

Department of Computer Science
National University of Computer & Emerging Sciences

Islamabad, Pakistan.

January 2020

Plagiarism Undertaking

I take full responsibility of the research work conducted during the Masters Thesis titled *Machine Learning Driven Load-Shedding in Parallel Complex Event Processing* . I solemnly declare that the research work presented in the thesis is done solely by me with no significant help from any other person; however, small help wherever taken is duly acknowledged. I have also written the complete thesis by myself. Moreover, I have not presented this thesis (or substantially similar research work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

I understand that the management of National University of Computer and Emerging Sciences has a zero tolerance policy towards plagiarism. Therefore, I as an author of the above-mentioned thesis, solemnly declare that no portion of my thesis has been plagiarized and any material used in the thesis from other sources is properly referenced. Moreover, the thesis does not contain any literal citing of more than 70 words (total) even by giving a reference unless I have the written permission of the publisher to do so. Furthermore, the work presented in the thesis is my own original work and I have positively cited the related work of the other researchers by clearly differentiating my work from their relevant work.

I further understand that if I am found guilty of any form of plagiarism in my thesis work even after my graduation, the University reserves the right to revoke my Masters degree. Moreover, the University will also have the right to publish my name on its website that keeps a record of the students who plagiarized in their thesis work.

_____

Qurat ul Ain Aftab

Date: _____

Author's Declaration

I, Qurat ul Ain Aftab, hereby state that my Masters thesis titled *Machine Learning Driven Load-Shedding in Parallel Complex Event Processing* is my own work and it has not been previously submitted by me for taking partial or full credit for the award of any degree at this University or anywhere else in the world. If my statement is found to be incorrect, at any time even after my graduation, the University has the right to revoke my Masters degree.

_____

Qurat ul Ain Aftab

Date: _____

It is certified that the research work presented in this thesis, entitled "Machine Learning Driven Load-Shedding in Parallel Complex Event Processing " was conducted by Qurat ul Ain Aftab under the supervision of Dr. Adnan Tariq.

No part of this thesis has been submitted anywhere else for any other degree.

This thesis is submitted to the Department of Computer Science in partial fulfillment of the requirements for the degree of Masters of Sciencein Computer Science

at the

National University of Computer and Emerging Sciences, Islamabad, Pakistan

January' 2020

Candidate Name: Qurat ul Ain Aftab Signature: _____

Examination Committee:

1. Name: Dr. Muhammad Asim Signature: _____
   Assistant Professor, FAST-NU Islamabad.
2. Name: Dr. Asma Ahmad Signature: _____
   Assistant Professor, FAST-NU Islamabad

Dr. Waseem Shahzad _____
Graduate Program Coordinator, National University of Computer and Emerging Sciences, Islamabad, Pakistan.

Dr. Kashif Munir _____
Head of the Department of Computer Science, National University of Computer and Emerging Sciences, Islamabad, Pakistan.

Abstract

With the advent of technology, Internet of Things has become quite popular in the world. Millions of sensor devices are connected, dealing with billions of incoming event streams originating from input sources. These massive event streams are processed by operators in Complex Event Processing to interpret particular situations. Complex Event Processing is a framework to process single or multiple events and to identify meaningful event patterns from multiple event streams. In the area of parallel CEP, related work has mentioned a few traditional approaches of operator parallelization, i.e. a key-based, a batch-based or a window-based. Operator parallelization reduces the input load through horizontal scalability. However, it has the significant drawback of correctness in high peak loads. At high peak loads, some essential events are randomly dropped at a single point in time. So the concept of load-shedding got highlighted. Through literature, it is known that all existing load-shedding approaches are limited to specific query operators and are not able to achieve correctness in abnormally high data rates, which is a prime requirement of a CEP system.
So, the challenge here is to decide where and how many events should get drop with minimal effect on correctness. To solve this problem we proposed a black box approach. In black box approach it does not matter what query operator is being used as only the inputs and outputs of the query operator are being observed. Deep learning models are applied to learn patterns from the observational data and the model predicts which events should get drop.

Acknowledgements

First, I would like to thank Allah Almighty for giving me the knowledge, strength and health to complete my research thesis. I would like to gratitude my thesis advisor Dr. Muhammad Adnan Tariq. The door to his office was always open, whenever I had a question about my research. His supervision constantly guided me in the right direction. I am grateful that he took me under his wing when I first approached him. Dr. Muhammad Adnan leadership has helped me grow into my potential and has opened my eyes to new stages of opportunity and strength.

I would like to regard, the honorable members of the panel, who refined my idea from the beginning meanwhile their quality suggestions improved my work.

Finally, I am grateful to my family and friends for giving me the support I wanted and constant motivation throughout my years of study and research.

Dedication

This research work is dedicated to sir Muhammad Zubair and all my mentors who have been unbelievably supportive through thick and thin. Without them I wouldn't be, where I'm today.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today is an era of Internet of Things in which billions of physical devices are connected to the internet. Internet of Things is an intelligent technology which includes sensing, information collection, information sharing, processing and intelligence.

1.1 Motivation

The motivation of this research is Complex Event Processing. An event is something that is happening in the external world. A complex event is a set of simpler events. A complex event is detected by analyzing the behavior of simpler events which are highly co-related. Complex Event processing detect composite events in time sensitive applications. For instance, Hygiene indicator is a time sensitive application in which immediate responses are required to prevent the spread of infectious diseases. In hygiene indicator, health worker exits patients room, took off surgical mask, did not sanitize and enter into other patients room is considered as composite event. Health-care worker exits the patient room, this activity is measured by a sensor reading and configured as "exit" event, took off the surgical mask referred to as "!wear Mask", did not cleanse the hands referred to as "!sanitize" and then entered into next patients room, an activity modeled as "enter". Such a sequence of event pattern i.e **SEQ(exit,!Mask,!sanitize,enter)** is detected by a CEP system which triggers when the hand hygiene rules are violated. The health-care workers are notified through some hand-ware device.(see the Figure 1.1).

In order to monitor the activities of all health care workers in a hospital environment, thousands of events take place at any point in time. So, a huge amount of data streams are continuously being generated which are difficult to tackle with traditional CEP systems. To process these large data streams, Parallel CEP systems are introduced. Parallel CEP systems provide support for many type of operators like (sequence, aggregate etc.). To efficiently process high data rate, operator parallelization techniques were introduced. Later, It was highlighted that there is a rapid increase in data rate at some point in time which randomly drops some necessary data. This random drop of data leads to correctness issues. Thus the concept of load shedding came into the limelight. All operator parallelization and load shedding techniques are discussed in literature review. (see section 3.1.1).

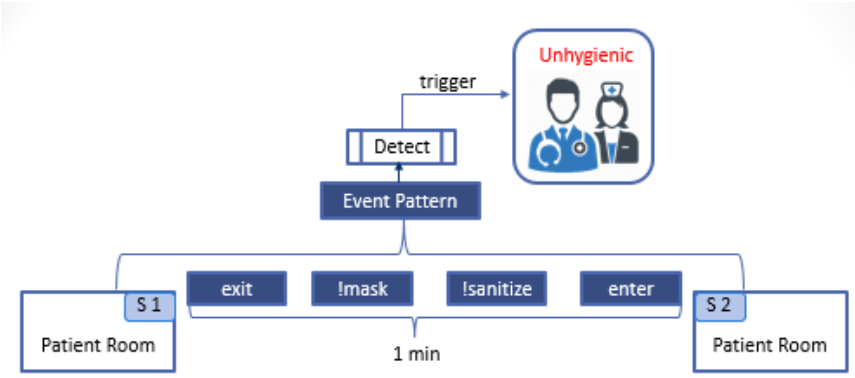

Figure 1.1: Health Indicator

1.2 Problem Statement

In situation aware applications like traffic monitoring systems, event detection mechanisms are used to detect complex patterns and process large streams of data. At certain times there might be a sudden fluctuation in workload and the already assigned instances are not capable to handle that rapid increase in data, so the time required to allocate a new instance will randomly drop some necessary events which will result in false negatives and false positives. The existing load shedding techniques of random drop, semantic drop and window drop had two major drawbacks i.e correctness and limited to specific operators. So, there is a demand to develop a general-purpose solution which can observe the performance of any operator i.e (sequence,aggregate etc.) and timely detect particular situations [1] [2] [3].
The problem statement in simple words is described as:

> Solving the problem of random drop at single point in time, to achieve correctness, in different application scenarios under abnormally high workloads, using deep learning techniques. Correctness here refers to false negatives and false positives.

In this thesis, we explore an approach for intelligently dropping incoming events and maximize the amount of detected complex patterns. Maximize means with minimum false negatives and false positives. We will use deep leaning models to predict the drop status of incoming events.
Following research questions are examined in this research:

1. Where to shed the load? (means load can be shed by dropping events before the events get processed in a CEP system).
2. How to shed the load? (means load can be shed by using deep learning models).

Previous load shedding techniques were operator specific and were not able to achieve correctness on sudden high data rate
The main objectives of our research is to achieve:

1. Little impact on Correctness
2. Operator Non Specific

The rest of the report is organized as follows. Chapter 2 describes an extensive background of CEP. Chapter 3 describes related work of operator parallelization and load-shedding techniques. Chapter 4 describes our methodology in depth. Chapter 5 discusses the experimentation and evaluations.

Chapter 2

Background

2.1 Information Flow Processing

Information flow processing is the mobility of information from sources to the consumers. In [4], traditional ways to process flows of information is explored. Traditional Information Flow Processing (IFP) engines were used to support situations like traffic, health, environmental monitoring or intrusion detection systems etc. All these time sensitive applications need to process information continuously and timely. Traditional DBMS cannot fulfill the necessity of timeliness as the query is run once to get a complete answer. This may take few hours to process data and return result which is not affordable in time-sensitive situations. Various Information Flow Processing systems such as data stream management systems (DSMS), database management systems (DBMS) , and publish-subscribe systems to complex event processing systems (CEP) are briefly discussed below. Data stream management systems(DSMS) belongs to data base management(DBMS) community. One of the major difference is that DBMS works on persistent data and DSMS works on transient data. Persistent data means information which is infrequently accessed and are not likely to be updated. Transient data means information which is continuously updated. Traditional publish subscribe systems speculate each event individually whereas complex event processing (CEP) systems consider composite events.

2.2 Complex Event Processing

In Introduction, we have already described some basic concept of a CEP system. In this section we will elaborate CEP system in depth. Basically, Complex event processing is an emerging innovative technology to filter and process event in real time scenarios like traffic monitoring , GPS navigators, weather forecasting etc. Complex Event Processing (CEP) systems track and analyze data streams generated from multiple sources to detect complex event patterns. Generally, Complex event processing systems filters low level information into a higher level information. (see figure 2.1).

In CEP, query languages like Esper, Snoop, SASE or TESLA are used to detect events. These languages include some language constructs like disjunctions, conjunctions, logical, sequences and negations to detect events.

CEP Architecture

Data parallelization framework consists of following components in its environment see figure 2.1).

Splitter

Splitter splits the incoming events into windows then assign these windows to different operator instances. Here the windows got open or close at the splitter level. Time based window gets open for a particular time and then get closed. In count-based windows. Certain number of events are processed then window got closed. Count window doesn't have any specified time limit. The difference between time based and count based windows are illustrated in 2.3. Window operator detects a pattern within a single window .After processing events in windows, an acknowledgment is sent to the splitter queue to discard the consumed events. A processed event is considered as consumed event.

Operator Instances

The operator instances process the windows in a parallel manner and detect event patterns.

Merger

All the outgoing events are reordered by the merger and the final output is produced.

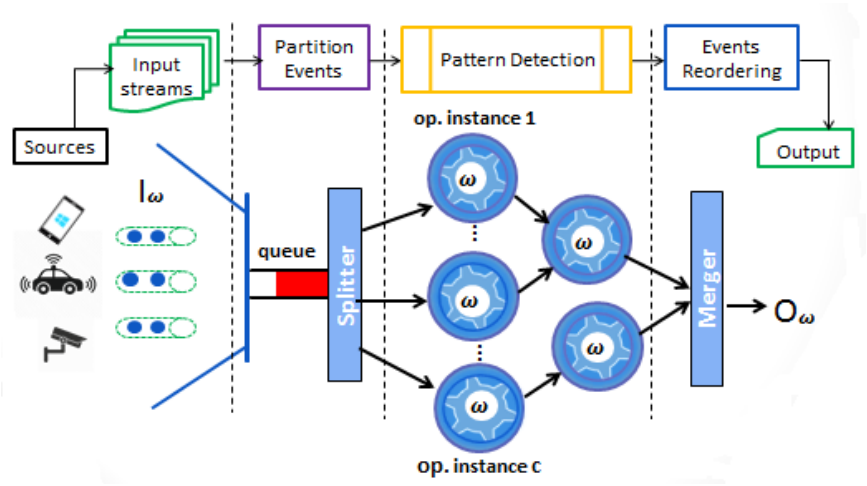

Figure 2.1: Complex Event Processing Framework

2.2.1 Example: Traffic Monitoring

Measure Vehicles Performance
In smart cities, automated traffic monitoring systems are deployed. On hilly areas there are dangerous steeps. Steeps are defined as a rise or fall from a certain angle. In case of a slope in a hilly area, vehicles brakes are highly essential because brakes are the important feature which can stop the car quickly and safely when required. CEP systems can be used to detect the performance of brakes by evaluating a particular pattern. CEP system proactively notify the drivers to avoid any calamity ahead.

This particular scenario is explained in simple words i.e. if the speed of the car exceeds a certain threshold and the brakes are applied at some time t ,a slope detected at some distance d and the weather is rainy. On this information CEP intelligently applies some decision rules, detects a complete pattern and depicts that the performance of the brake is poor and the car may skid off the road (see Figure 2.2). Event streams captured from the sensors (speedometer, brake, GPS, Thermocouple, Barometer) are sent to the $\omega speed$, $\omega brake$, ωGPS, $\omega temperature$ and ωair operators These operators extract speed, time, brake values of distinct cars. Weather situation is measured by taking temperature and air pressure readings. Predicates are applied on each operator from low level operators up to the higher level. Complex patterns are detected and then the useful information is extracted.

Operator Execution Queries
Vehicle exceeds a certain speed, this event is measured as "!speedLimit", Brakes Applied referred to as "!brakePerf" event, Slope is detected is referred to as "slopeAhead" event, weather status modeled as "rainy" event, then the sequence of complex pattern will be
Seq("!speedLimit", "!brakePerf", "slopeAhead", "rainy")
which triggers when the performance of the brakes are poor (the symbol ! means negation). These types of complex patterns can be written in query languages like Esper, Tesla, Snoop or SASE.

Vehicles Brake Performance

```
PATTERN (A, B, C, D)
DEFINE
A AS A.Type="speedLimit" and "time"
B AS B.Type="sensorValue" and "time"
C AS C.Type="location"
D AS D.Type="weather"
within ws time units From A
```

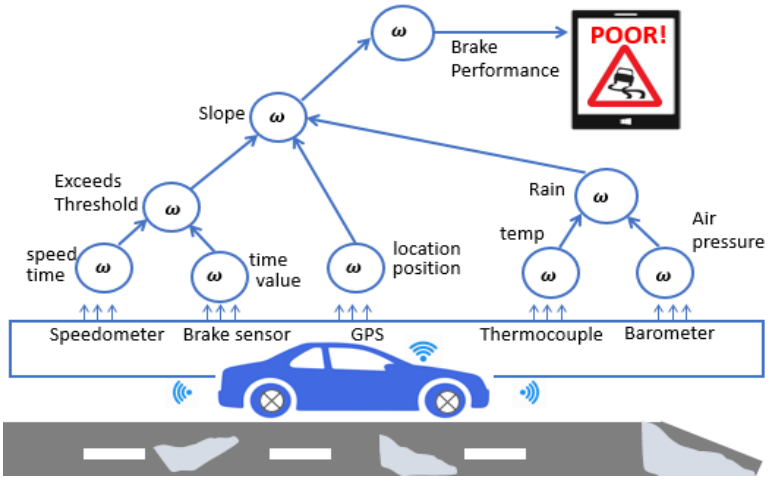

Figure 2.2: Traffic monitoring: Surveillance on hilly areas. In this example the hierarchy of operator graph shows, how low-level information from sensors are filtered into a higher-level information

2.3 List of Operators

The table demonstrates various operators.

Table 2.1: List of Operators

Sr#	Operators	Types
1	Flow Management	Join Union Filer
2	Bag	Itersect Uion Dplicate GoupBy
3	Aggregate	Sum Count Avg
4	Logic	Conjunction Disjunction Negation
5	Single Item	Selection Projection Renaming
6	Sequence	Seq
7	Drop	Drop
8	Window	win

2.4 Window Specifications

In CEP, the window specification defines the size of the windows and their slide time. Window slide demonstrates when to open the next window, for example if window slide is 1 min then after every 1 min next window will get executed.

Two types of windows are mainly discussed in literature review of CEP i.e time based and count based. Time based windows gets open for a particular time to consume events and then get closed where as count-based windows contains certain number of events and then get closed. By processing count based and time based windows with same slide time, lets suppose 1 min, it is analyzed that both have different results.To get the same results, the parameter evenpersec is set. Event patterns are detected in the window operator (see figure 2.3).

2.4.1 Example:Tweets

The below mentioned query shows the tweets which are received more then 10 times in last 10 sec. This query count tweets for different topics.

```
Select topic,Count(*) from twitterStream TIMESTAMP BY CreatedAt
GroupBy topic, SlidingWindow(second,10) having count(*) > 10;
```

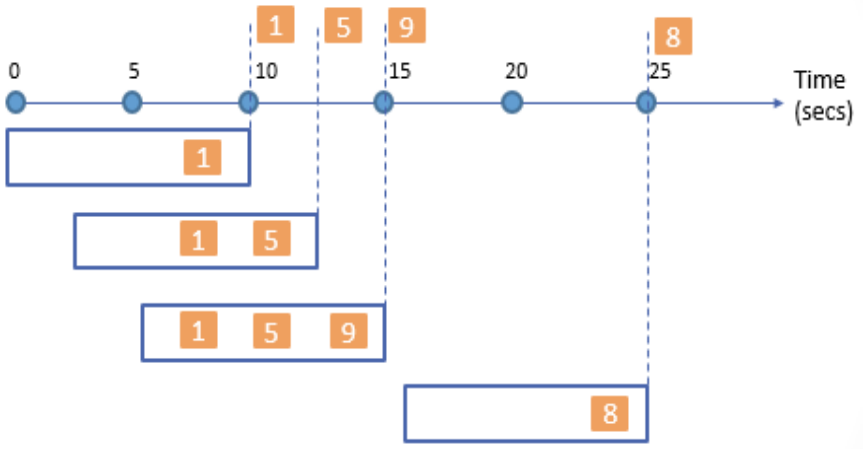

Figure 2.3: 10 seconds Window Slide

2.5 Consumption Policy

Consumption policy guides what will happen to an event once it is used in the generation of complex events. There are two possibilities; First, the event can be reused in the generation of other complex events. Second, the event is removed from the window. We will use the later one [5].

For example, let us assume, if we execute the operator to process events. Three events A, B and C occur in a scope of 1 min window. In case of a pattern match, consumption policy consumes all processed events and discards all unprocessed events.

2.6 Deep Learning Model

Deep learning trains machines to learn and think like a human intellect. We will use deep learning models because they are most appropriate when dealing with big data. Big data is defined as extensive data sets that may be analyzed computationally to recognize patterns and trends. Deep learning algorithms follow the brain simulations which are generally known as neural networks. We used different types of neural networks, i.e. feed-forward and recurrent neural network. In this thesis, the neural network model is used to predict drop status of the next incoming event, whereas rec-current neural network keeps the memory and is used to predict which events in a sequence should get a drop in future.

2.6.1 Feed Forward Neural Network

Feed forward neural networks are used to train observational data. The brief history of artificial neural network is discussed in [6]. Basically, Artificial neural networks (ANN) are similar to biological neural networks. A biological neural network consists of neurons. Each neuron is responsible for transmitting and processing information. In ANN, a neuron is a mathematical unit used for computations. Basically, ANN learns the patterns through experiences as the human brain does. It means the neural network will be able to generate new outcomes based on the previous knowledge it has learnt from historical data. An ANN architecture mainly consists of three layers input layer, hidden layer and output layer (see figure 4.4). Each input value x1,x2..xn is multiplied by initial weight values w1,w2..wn and computes a sum. On each computed sum an activation function sigmoid,reLu or tanh etc. is applied (see below-mentioned equations). Sigmoid Activation function normalizes the values in a particular range between 0 and 1 (see equation (1), (Eq: 2)).

$$z = x_1 * w_1 + x_2 * w_2 .... + x_n * w_n + b * 1 \quad \text{(Eq: 1)}$$

$$a_{out} = sigmoid(z)$$

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \quad \text{(Eq: 2)}$$

The output from the previous layer is fed into the next hidden layer. The hidden layer resides between the input and output layer. The output layer generates the results. Then the difference between the actual and predicted result is calculated to get the error value. The error is minimized by iteratively updating the weights to a particular neuron where the error is acceptable. There are various loss functions to minimize the error rate. add reference. The architecture of a feed forward neural network is displayed in figure 2.4.

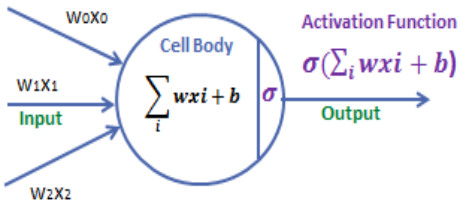

Figure 2.4: Artificial Neuron

Chapter 3

Literature Review

In this section, we briefly discussed state-of-the-art parallel CEP Systems, explored techniques to timely detect event patterns on single as well as cluster of machines. We examined various load-shedding techniques in the domain of stream processing as well as Complex Event Processing. We did a comparison between different approaches and examine whether each former approach performed better then the latter.

3.1 Parallel CEP

Parallel CEP systems are developed to detect events continuously. Recently, there have been many frameworks put forward for event detection on large data streams. Each of these frameworks follows a different technique. As these frameworks are not appropriate when there are high fluctuating work loads in an IoT environment, so number of techniques have been followed to achieve degree of parallelization.

3.1.1 Operator Parallelization

High event rates generating from input sources demand the need of scalable CEP operators. However, sequential operator techniques are not smart enough to utilize the multi-cores and elastic cloud resources. Therefore, there was a need to increase the operator parallelization degree to achieve scalability of the CEP. In the following,operator parallelization methods are described. Two major techniques are briefly analyzed: Task parallelization and data parallelization. Various techniques are illustrated in a tree like structure in figure 3.1

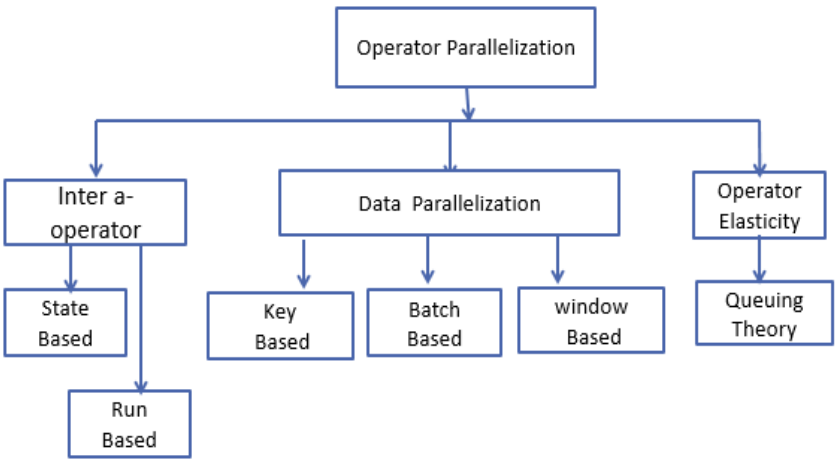

Figure 3.1: Operator Parallelization

1. **Intera-operator parallelization**

   In [7], author proposed a parallelization framework for stateful stream processing operators. A stateful operator saves incoming information items as states. The operator state is limited to time and scope. In intera operator parallelism, state based approach represents pipelined parallelism i.e output of one operator is input to another operator in sequence. Each processing unit in a CEP architecture has a finite state machine corresponding to a pattern variable. CEP architecture is divided into three parts, Input Handler, Pattern Evaluator, and Match Output Handler. Pattern evaluator is the main component of CEP with multiple processing units and has the highest computational cost. State based approach has several potential problems i.e. communication overhead, replication of input events, query dependence, load imbalance and limited scalability. Moreover, transfer of partial matches between multiple processing units is another overhead. These drawbacks lead to the thought to evaluate (events, partial matches, states) on the same processing unit referred to as run based approach.

   In run-based parallelism, pattern matching is performed on on each processing unit. The cost of carrying partial matches between the processing units is avoided. RIP approach works on a round robin algorithm. Authors implement operator query as finite state machines. These finite state machines are mapped onto parallel processing units of multi core CPU architecture. To avoid load imbalance, an input stream is split into batches. Each batch is assigned to a single processing unit. Authors focus in this research was throughput.

   The proposed run based approach achieved linear scale up, increased throughput and is query independent.The results showed that state based technique is query dependent and ideal throughput couldn't be achieved where as run based approach is more vibrant to different queries and is query independent. All technical details are correct and can be verified by implementation and statistical analysis (see figure 3.2)

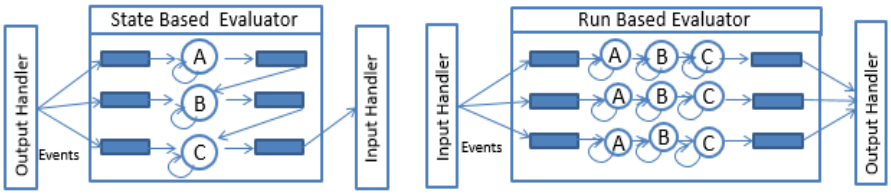

Figure 3.2: Task Parallelization

2. **Data parallelization**

   In data parallelization [8][9] the input streams are split into partitions. These partitions are processed on homogeneous computing nodes. Figure 2.1 illustrate data parallelization framework. This framework contains three main components a splitter, multiple operators and a merger. The critical question answered in this research is how the input event streams are split to detect event patterns consistently. The approach of data parallelization has three major drawbacks; load balancing, expressiveness and scalability. Three partitioning models are discussed,key based , batch based and pane-based

(a) **key Based**

In [10], Key-based partitioning splits the event streams into data chunks that produce intermediate key-value pairs. There are limited keys available for each operator instance. For example, to detect a stock pattern in the stock exchange market, there are limited keys available to the number of distinct stock symbols. If the keys are unevenly distributed among operator instances then there is another draw back of load imbalance. Sometimes,there are no common keys available to group the events to various operators.The limitation of this approach is that no temporal relations are handled for situation aware applications.

(b) **Batch Based**

Batch-based partitioning in [7], split the input event streams into batches. These batches are formed depending on the size of the query pattern. This approach is not appropriate to detect patterns of an unknown size. The batch processing may produce results after 2 hours which are not affordable in time sensitive applications.

(c) **Window Based**

The input stream is split into partitions called windows. The concept of windows is discussed in detail in [9]. There are various types of windows discussed in the literature review on CEP i.e time based,count based and sliding windows. Each window is assigned to a particular operator instance. These operator instances perform parallel processing and detects event patterns. The major drawback of window based splitting is communication overhead as discussed in [11]. To eliminate this overhead the concept of window batching came into existence in which overlapping windows were compact in a batch and were assigned to a single operator instance.

(d) **Pane Based**

Pane-based partitioning split the input event streams into window panes. This approach is discussed in detail in stream processing systems [9]. For instance, window is divided into 6 fragments, each fragment is of 10 seconds, that means each window is opened for exactly one minute for computations.

3. **Parallelization Degree Adaption**

In [8], pattern sensitive stream partitioning models are proposed to ensure parallelization and achieve the parallelization degree using queuing theory. Queuing theory observes the buffer fill rate and operators processing time. It proactively decides if an additional instances are required to balance the workload among operator instances thus minimizing the unlimited buffering delays. For instance in a traffic monitoring scenario, workload increases in rush hours at day time as compared to mid night, so the number of instances allocated or deallocated are according to the workload fluctuations. At high data rates, operator is overloaded which may result in unlimited buffering delays. Delays in event detection are caused due to communication overhead, processing and buffering limit of events. Future work will aim at developing a latency model for operators to reduce communication overhead.

3.2 Loadshedding

In IoT, there are certain times at which event arrival rate is too high that it exceeds the systems capacity resulting in dropping some necessary events. These necessary events are essential in time sensitive applications to detect a particular situation.

To the best of our knowledge, very little research is done in this domain. Aurora [1] was the first system to perform load shedding on flurry input rates. Aurora is a relational stream processing engine. In Aurora, drop operators were automatically inserted in to the executing query network. Load Shedding is basically eliminating of surplus load from the systems. Three types of drop operators are mainly discussed in literature i.e random drop, semantic drop and window drop see the figure 3.3. All these load-shedding strategies have major drawbacks of correctness which is a prime requirement of any CEP system.

1. **Random Drop**

   In [12], authors suggest load-shedding under CPU bound and memory bound resource constraint. During times of peak loads, the capacity of the system is overloaded so all input events cannot be periodically processed under resource constraints. As a result, some of the incoming events are discarded and the CEP system generate subset of results. Two types of shedding mechanisms are discussed in this paper i.e integral and fractional load-shedding. In integral load-shedding certain types of evens are eliminated whereas in fractional load-shedding segment of event types are randomly selected and discarded.

   In [1], the load shedding road map (LSRM) data structure for random load shedding is introduced. LSRM make decisions like when and how much load to shed. This system haphazardly select fraction of tuples from the input streams and then drop it. LSRM contains precomputed drop insertion plans. This approach can work only when the input rate of events is too slow. Thus in [12] [1], the authors claimed that by randomly dropping a percentage of tuples will result in inconsistent data which is not tolerable in situation aware applications.

2. **Semantic Drop**

   In [1], authors used semantic approach which filter tuples on the basis of their content. Data which contains low value content is dropped. Analyzing the significance of the content is another research challenge. If drops are inserted into the query operators. The output is guaranteed to be a subset. In this paper, LSRM data structure is used

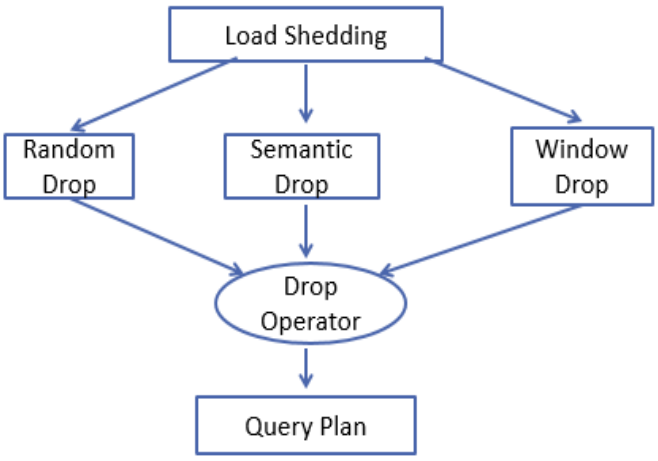

Figure 3.3: Load Shedding

for semantic drop. Their objective is to filter out data values with lowest utility.The focus of this research was to save CPU processing cycles by applying precomputed drop insertion query plans. This approach is limited to filter operator and will not work under busty input rates.

3. **Window Drop**: The authors introduced a window drop load shedding in data stream processing as well as in complex event processing. In [13] the major goal of the author is to develop load shedding algorithms which will reduce the overload and will produce subset of original results. The load shedding strategies are applied on queries consisting of aggregate operators. A new window drop operator is introduced which consist of window size and window slide properties. This drop operator probabilistically decides which windows to drop. Precisely, the downstream operators in a query network analyze the window specifications and decides whether to keep the window or discard it. One strategy is to shed the tuples after applying aggregate operators. Another strategy is to drop a complete window at early point in a query network. Former will not save from computation cost but will produce subset results while later will require less computation but will produce subset results. To handle multiple aggregates two types of arrangements are used i.e pipeline and fan-out.

   In [3], authors did the latest research in window drop load-shedding in the context of Complex Event Processing. In this thesis, authors introduced a strategy to evaluate both latency and quality of windows. Authors faced three main challenges; one was to determine the quality and processing latency of windows. They developed an estimation model which predicts how many complex events a window is expected to constitute. For this challenge, the weighted average strategy and Markov reward model is computed. The weighted average strategy will analyze the previous processed events of the operator instance and will produce an average value. This approach does not consider variations in distributions of events. Markov reward process probabilistically moves from one state to another state to detect a sequence of events. For each state markov computes the cost and reward of each state after processing number of events. The state transitions can be observed at any point in time. The second challenge was to design a window dropper algorithm that will detect complex events and will drop a window in case of overload. Window dropper probabilistically drop windows by calculating processing cost and complex patterns in each window. The third challenge was to save the processing cost of a window so that latency bound does not exceed a certain threshold. For this challenge, residual latency estimator was developed. Residual latency estimates analyze systems current load and decides which windows should get drop. The authors named this approach the white-box approach, which in real life is not that easy to implement and is limited to a specific operator.

3.2.1 Additional Literature

1. **Reducing Communication Overhead in Window Based Partitioning**
   In [11], latest research on window partitioning by minimizing communication overhead is done. In past, to expand the throughput of an operator, multiple windows were appointed to different operators i.e. non distinct operators. These operators process windows in parallel. Events that are part of different overlapping windows exist in multiple operator instances which induced operator processing load,latency and communication overhead . In this paper, authors goal is to allocate overlapping windows to the same operator instance. This challenge cannot be overcome by traditional reactive methods. So model-based batch scheduling controller is developed on prediction. This approach saves bandwidth, reduce latency bounds in operator instance and showed that the controller batches windows even at busty workloads.

2. **Elastic Scaling for Data Stream Operators**
   High Volume data streams generated from multiple sources need efficient parallel processing capabilities to integrate with elastic features. In [14] proposed an elastic strategy based on a predictive model. This model takes proactive optimal scaling decisions and reduced the latency. Scaling decisions should be taken by the operator automatically according to the workload fluctuations monitored at run-time. They evaluate the effect of re-configurations in minimizing the latency. After experimentation the results showed that this approach takes necessary re-configurations while providing reduced resource consumption. This approach is not developed in real time environment. In future the authors wants to integrate their approach with real time stream processing frameworks like Apache Storm.

3.2.2 Comparison Metrics

Table 3.1 demonstrates the comparison matrix of some of the significant papers that are relevant to our research.

Table 3.1: Comparison Metrics

Ref	Author	Approach	Architecture	Tool	TP	Lat	Scal	Corr	Cost	Para	OS	LS
[4]	Margara	Complex Event Processing	Multi-core Multi-Machine, Cluster	Esper	✗	✗	✗	✗	✗	✗	✗	✗
[7]	Dean	Map Reduce (Key Based)	Multi-core machines, multi-machine clusters	Hadoop	✓	✓	✓	✗	✓	✓	✗	✗
[5]	Balkesen	Intra operator Parallelism (State Based Run Based)	Multi-core CPU	Esper	✓	✓	✗	✗	✗	✗	✗	✗
[15]	Balkesen	Data Parallelism Batch based	Multi-core CPU	Esper	✓	✓	✗	✗	✗	✗	✓	✗
[1]	Mayer	Parallelization Degree Adaption Queuing Theory	multi-machine clusters	Snoop CQL	✓	✗	✓	✓	✓	✓	✗	✗
[8]	Matteis	Elastic Scaling	Multi-core machine, multi-machine cluster	-	✓	✗	✓	✓	✓	✓	✗	✗
[9]	Tatbul	Semantic Load Shedder (Data Stream Manager)	Multi-core CPU	Aurora	✓	✗	✗	✗	✓	✗	✓	✓
[11]	Rivetti,	Load shedding Advance Planning (Dist Stream Processing)	Multi-machine Cluster	Aurora	✓	✗	✗	✗	✓	✗	✓	✓
[10]	Babcock	Window aware load shedding (aggregation queries)	Multi-machine Cluster	Aurora	✓	✗	✗	✗	✗	✗	✓	✓
[12]	Albert	Window Drop Load Shedding (white box)	Multi-machine Cluster	Markov Esper	✓	✓	✗	✗	✗	✓	✓	✓

Key:
The above mentioned table shows the comparison between the following parameters:
Throughput=>TP
Latency=>Lat
Scalability=>Scal
Correctness=>Corr
Operator Parallelization=>OP
Operator Specific=>OS
Load Shedding=>LS

Chapter 4

Methodology

In our methodology, we will follow a black-box approach. Black-box means we will not explore the internal working of the operator. We will only observe the input and output of the operator. By observing the input and output, we will know what type of events are being generated in case of complex events. For example, in case of a sequence operator, we will know what type of sequences are getting generated by the operator. The events filtered by the operator are called consumed events, and the events which are left behind are called unconsumed events according to the consumption policy discussed in section 2.5.

The below-mentioned figure 4.1. clearly describes the overview of our methodology. The black box could be any operator, i.e. (aggregate, sequence, range .etc). The events in the input queue are ABCD, and the events in the output queue are ABD. The events ABC are filtered by the operator (for e.g sequence operator) as seen in the output queue. Both the input and output events are merged to form a dataset. This dataset is then ready to get trained by the neural network. The neural network predictive model learns from the actual data and predicts which next incoming event should get the drop with minimal impact on correctness. This predictive model is named as Load shedder. By correctness, we refer to false negatives and false positives. False negatives occur when an event is processed in actual data but predicted as the drop. False-positive occur when an event is the drop in actual but is predicted as processed.

In order to solve this problem, first, we need to generate data as mentioned in section 4.1 , then we require deep learning models, i.e. neural networks as addressed in section 4.4. Steps of the proposed methodology are clearly mentioned in figure 4.2.

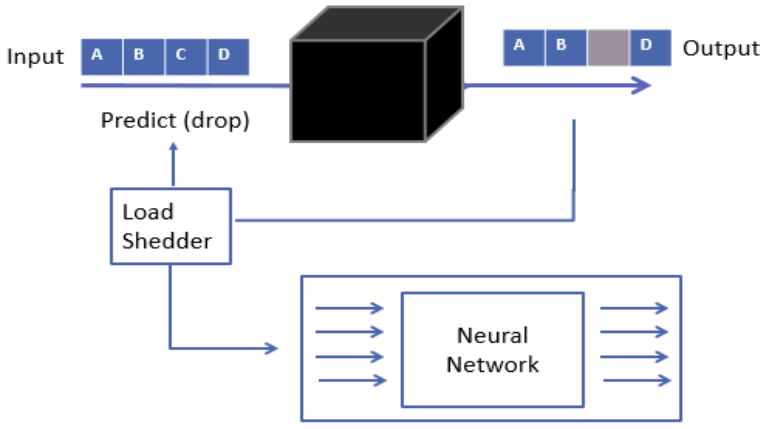

Figure 4.1: Black Box

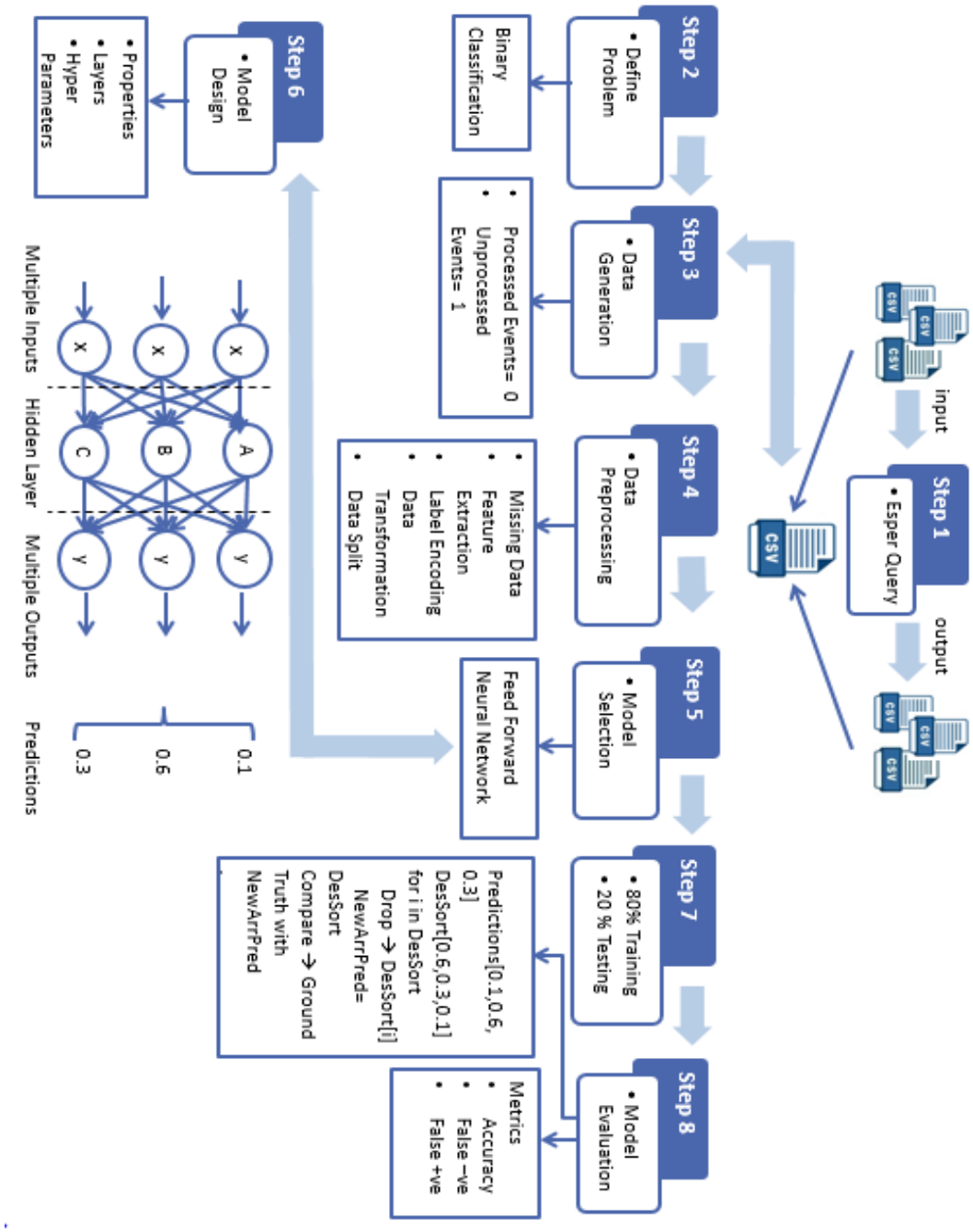

Figure 4.2: Steps of the Methodology

4.1 Data Generation

The data generation concept is illustrated in figure 4.3 from step 1 to step 3. This diagram displays how input and output of operators are observed and merged. In data generation ,we will take one real dataset and one synthetic dataset. We focus mainly on two application scenarios. The first scenario is a stock exchange which is a real dataset, and the second scenario is overtaking in traffic monitoring systems which is a synthetic dataset.

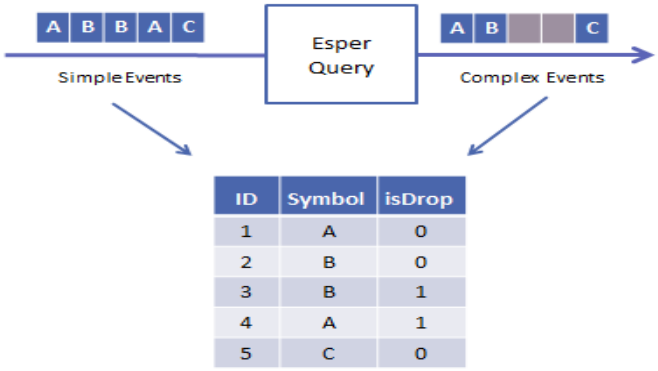

Figure 4.3: Data Generation flow diagram (0 represents processed events and 1 represents unprocessed events)

4.1.1 Stock Exchange

A stock exchange is a market where stocks, shares and bonds are bought and sold. Stock prices change every day for example if more people demand to buy a stock then its supply, the stock price rise and if more people want to sell their stock then purchase it, the stock price falls. Thus the Stock Exchange market keeps track of the rise and fall of stock prices.

Real Data:

In stock exchange scenario we will use the following dataset and queries:

1. First, a real-world New York stock exchange (NYSE) dataset is taken from Kaggle and is used for technical analysis by applying queries in NEsper. This dataset contains price information of stocks from 2010-2016. The price.csv file consists of six attributes (date, symbol, open, close, low, high, volume) and total 851265 tuples. By executing different queries, we can generate different types of patterns.

Table 4.1: Attributes of Price.csv

Sr#	Attributes	Description
1	symbol	Company Symbol
2	open	The opening market price of the particular symbol
3	close	The closing recorded price of the particular symbol
4	high	The highest market price of the particular symbol
5	low	The lowest market price of the particular symbol

2. In our experimental evaluation, we will apply various types of queries on this dataset. We will implement various queries in event specification language known as esper. For stock exchange dataset query 1, 2, 3, 4 can be applied. These queries are picked from related work. [3].

(a) Q-1, opens a window with a scope of 1 min when an event A occurs followed by an event B.

```
-------------------------------------------------
Q-1: Detect distinct sequence in a Pattern(A B)
-------------------------------------------------
        PATTERN (A B)
        DEFINE
        A AS A.type = A
        B AS B.type = B
        WITHIN 1 minute FROM A.timestamp
```

(b) Q-2, detect events and compare price of current or next event with a price of previous event.

```
 -------------------------------------------------
 Q2: Detect Complex Pattern
 -------------------------------------------------
        PATTERN(A B  C)
        DEFINE
        A AS (A.price < 70) ,
        B AS (B.price > PREV(B.price)
        AND B.price > A.price) ,
        C AS (C.price > PREV(C.price )
        AND C.price > 130)
```

(c) Q3, detects a complex event when a change occurs in the stock symbol price between lower and upper limit. Upper and lower limits are used to define the average pattern size. "Kleen+" shows that many events can match. "WITHIN" clause specifies a time period.

```
-------------------------------------------------
Q-3: Detect stock symbols closing price in a range
-------------------------------------------------
        PATTERN (A B+ C)
        DEFINE
        A AS (A.closePrice < lowerLimit),
        B AS (B.closePrice > lowerLimit
        AND B.closePrice < upperLimit),
        C AS (C.closePrice > upperLimit)
        WITHIN ws events FROM every s events
        CONSUME (A B+ C)
```

(d) Q-4, displays the stock rising pattern. RE and MLE are stock symbols. Stock symbols opening and closing prices are compared in this query.

```
-------------------------------------------------
Q-4: Stock Rising Pattern
-------------------------------------------------
        PATTERN ( MLE RE1 RE2 ... REq)
        DEFINE MLE AS ( MLE. closePrice> MLE. openPrice),
        RE1 AS ( RE1. closePrice> RE1. openPrice),
        RE2 AS ( RE2. closePrice> RE2. openPrice),
        .. REq AS ( REq. closePrice> REq. openPrice)
        WITHIN was events FROM MLE
        CONSUME (MLE RE1 RE2 ... REq)
```

4.1.2 Traffic Monitoring

In Traffic Monitoring scenario of an expressway. There exist a few areas where overtaking is a ban, there are two cameras (Src1 and Src2) deployed at the beginning and end of the no-passing zone area (L1 and L2). When a vehicle "a" crosses location L1 after some time another vehicle "b" enters location L1 and then crosses location L2 before "a". This case shows that "b" has overtaken "a" and thus transgress the traffic rules. The below-illustrated figure 4.4 is taken from [8].

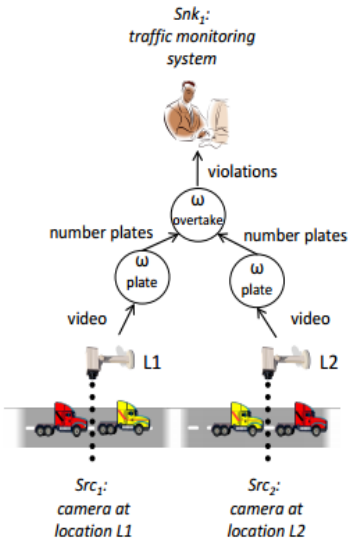

Figure 4.4: Vehicles Overtaking

Synthetic Data:

In vehicles overtaking scenario, we used the following synthetic dataset and queries.

1. We will generate dataset synthetically by using Poisson distributions. Poisson distribution is typically used to model the number of times an event occurs in a time interval. The average number of events in an interval is denoted by lambda. The equation mentioned below calculates the probability of k number of events in a specific interval (see formulae mentioned below). For instance, the number of cars entering location L1 and exiting location L2 in a time interval is considered as a Poisson process.

$$P(k \text{ events in interval}) = e^{-\lambda} \frac{\lambda^k}{k!}$$

2. The synthetic dataset of traffic monitoring system contains information of vehicles like their number plates and crossing locations. The overtaking.csv file consists of three attributes (number-Plate, type, time) and total 100000 tuples (see table 4.2).

Table 4.2: Attributes of overtaking.csv

Sr#	Attributes	Description
1	number_plate	Vehicles number plate
2	type	Entering location L1 at time t1 and exiting location L2 at t2

3. In our experimental evaluation, we will apply the sequence operator on this dataset to detect overtaking. We will use esper to implement this query. Previously, this query was written in snoop and is picked from [6]. In Q5, Aperiodic means events not occurring at regular intervals. Event B occurs between two complex events A and C.

```
--------------------------------------------------
Q5: Detects a particular sequence
--------------------------------------------------
        Aperiodic(A;B;C) with
        A -> <plate=a, type=L1>,
        B -> Sequence (<plate=b, type=L1> ; <plate=b, type=L2>),
        C -> <plate=a, type=L2>,
```

4.2 Data Preprocessing-Step[4]

Data preprocessing key points are mentioned in step 4 of figure 4.3. Each key point is discussed in detail below:

1. **Missing Data**

   For data preprocessing, in stock exchange and overtaking data, first we will verify if there exist any missing values in our dataset. If there exist any missing values in our data we will remove that tuple.

2. **Feature Extraction**

   In the case of the stock exchange data, we extract symbol feature and drop all other features i.e (open, close, low, high, volume). In the case of overtaking data, we extract number plates, type location and drop the timestamp attribute. The reason to extract a few features and discard others depends upon the type of operator being executed.

3. **Label Encoding**

   In stock exchange data, the stock symbol is in text form and in overtaking, number plate and type location is in text form. Machines don't understand the text data, so we will encode the data by using LabelEncoder class from scikitlearn library. The figure 4.5 clearly explains the meaning of encoding.

4. **Data Transformation**

   We will transform the shape of our data from one structure to another structure because we want multiple inputs and outputs according to the specified size of data window. For example, for window size 10 we will transform 10 events into 10 inputs. The figure 4.6 gives an idea of data transformation strategy.

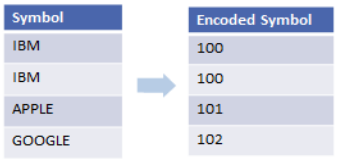

Figure 4.5: Encode the text data

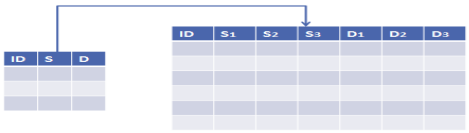

Figure 4.6: Transform multiple events into windows

5. **Train Test Split**

   In this step the data is split into training and test set. General rule of thumb is to allocate 80 percent data for training and 20 percent data for testing (see figure 4.7). We will import test_train_split from model_selection library of scikitlearn.

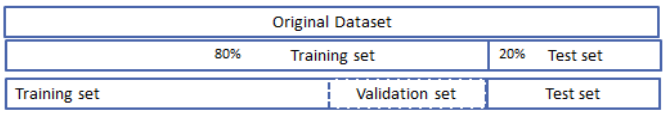

Figure 4.7: Data Split

4.3 Design Neural Network

We will design a Neural Network, the background of artificial neural network is discussed in section 2.6.1. The figure 4.3 at step 5 to 6 briefly illustrates the design of neural network. The neural network learns from observational data i.e training set and predict the drop status of an event.

1. **Network Architecture**

   We will give multiple inputs to the neural network model and corresponding to each input, an output is generated. Input events refers to each feature of the input data and output refers to, whether that event should get drop or not. The figure 4.9 illustrates the architecture of a neural network.

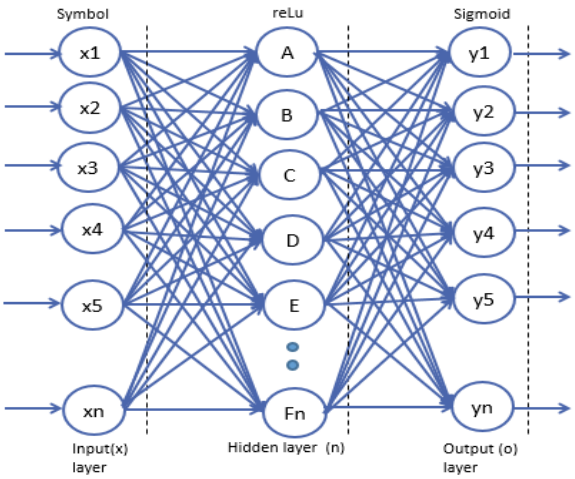

Figure 4.8: Feed Forward Neural Network

2. **Model Layers**

   The neural network illustrated above consist of three layers. First layer is the input layer. x1,x2,x3 are input features in the first layer and y1,y2,y3 are predicted outputs in the last layer. In the case of stock exchange, the inputs to the neural network are encoded symbols. In the case of overtaking scenario, the inputs to the neural network are encoded vehicles number plates and their crossing locations.

   The hidden layer resides between the input and output layer. It consist of number of neurons. Neurons are mathematical units. These units perform calculations on the weighted inputs and generate net input which is then added to produce the actual output with an activation function. There are 500 unique stock symbols in the dataset. So we decided to set 500 neurons in our hidden layer and in case of overtaking scenario, we reasonably decided to set 50 neurons.

   The output layer is the last, and it generates the predicted value. The output layer predicts whether an event should get a drop or not in case of binary classification .i.e (1/0). 1 represents the event should get drop and 0 represents that event should not drop. (see table 4.3).

Table 4.3: Model Layers

Sr#	**Layers**	
1	Input	Encoded Symbols
2	Hidden	# of unique symbols
3	Output	Binary

3. **Model Properties**

We will build a sequential model by using Keras library. Keras is one of the most widely used library because it is user-friendly, and it runs on top of Tensor-Flow. The Sequential() function, sequentially add layers in a model. Three types of layers are built in this model. The input, hidden and an output layer. The first input layer is flatten. The flatten layer transforms the 2-Dimensional data into a single vector. This vector transmits values into a dense layer. The dense layer is fully connected. The last layer is the output layer which is also a dense layer. The below-mentioned table illustrates the properties of a model in a very concise way (see table 4.4).

Table 4.4: Model Properties

Sr#	**Model Properties**	
1	Model Type	Sequential()
2	Libraries	keras, numpy, panads, sklearn, Matplot lib
3	No of Layers	3
4	Layer Types	Flatten, Dense

4. **Model Hyper Parameters**

We decide hyper-parameters before training a model, i.e. no of layers, learning rate, optimizer etc. as mentioned in table 4.5.

Loss function is very important while designing a neural network. In the case of binary classification, we will use binary cross-entropy which is widely used (see (Eq: 5)). In this loss function, y is the label (1 for drop events and 0 for processed events), and p(y) is the predicted probability of the event dropped for all N number of events. In this formulae, for each drop event (y=1), it computes log(p(y)) to the loss, which means the log probability of the event being drop. Conversely, it computes log(1-p(y)) which means the log probability for each event being process (y=0).

$$H_p = -\frac{1}{N}\sum y_i.log(p(y_i)) + (1-y_i).log(1-p(y_i)) \qquad \text{(eq: 5)}$$

Table 4.5: Model Parameters

Sr#	**Parameters**	
1	Loss	binary_crossentropy
2	Optimizer	adam
3	Activation Function	ReLu, Sigmoid
4	Learning rate	0.01
5	Batch Size	100
6	Epochs	100

4.4 Predictive Load Shedder

The purpose of this research thesis is to reduce the load by dropping events with minimal impact on correctness. Correctness refers to false negatives and positives. So after data training and validation. We randomly select 1000 samples from the data frame. The model.predict function predicts the probability for each event in a window. We observed the correctness by dropping events upto 25 percent from a window of size n. Step 8 in figure 4.3, describes the key points of load shedding mechanism.

The drop strategy is explained in illustration (see figure 4.9). This illustration displays the impact on false negatives and positives after dropping one, two or three events. The higher the probability of an event, the higher it has a chance to get a drop. For example, A4 has highest probability of 0.7 so it is the first drop candidate. B1 has the second highest probability, so it is the second drop candidate. By comparing the Ground truth with our predictions , we keenly observed the rate of change in false negatives and positives see table 4.6. This table shows if the complex event exists in ground-truth but do not exist in prediction then a false negative occurs. If the complex event exists in prediction but do not exit in ground truth then false positive occurs.

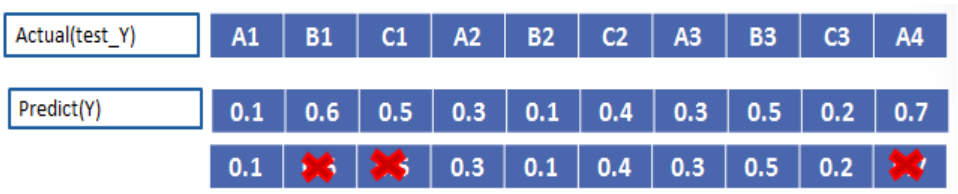

Figure 4.9: Data Events from a window

Table 4.6: Calculating false -ve and +ve,
By comparing ground-truth with prediction

Drop	Total Complex Events	False -ve	False +ve
0	6	0	0
1	6	0	0
2	5	1	0
3	5	1	0

5. **Metrics** Model evaluation parameters are discussed at step 9 in figure 4.3.

   We will use accuracy and typically the false -ve and +ve metric to evaluate our models performance. Accuracy formulae is simply a ratio of correctly predicted observations divided by total number of observations (see equation (Eq: 3)). For example, if the number of test samples is 2000 and model classifies 1952 of those correctly, then the model's accuracy is 97.6 per cent. The actual formulae of accuracy is mentioned below:

$$Accuracy = TP + TN/TP + TN + FP + FN \quad \text{(Eq: 3)}$$

   There are 4 important terms : TP: True Positive TN: True Negative FP: False Positive FN: False Negative

   (a) True Positives (TP): The cases in which the model predicted YES and the ground truth was also YES.

   (b) True Negatives (TN): The cases in which the model predicted NO and the ground truth was also NO.

   (c) False Positives (FP): The cases in which the model predicted YES and the ground truth was NO.

   (d) False Negatives (FN): The cases in which the model predicted NO and the ground truth was YES.

   We will consider both actual and predicted events of each window. The formulae to calculate the occurrence of false negatives and false positives after dropping events is mentioned below.
   TT-> Total number of complex events in actual window and in predicted window.
   FN-> Complex events that occurs in actual but do not exist in prediction.
   FP-> Complex events that occurs in prediction but do not exist in actual

   . The following equations (Eq: 4) and (Eq: 5) computes False -ves and +ves.

$$\sum \frac{FN}{TT} * 100 \quad \text{(Eq: 4)}$$

$$\sum \frac{FP}{TT} * 100 \quad \text{(Eq: 5)}$$

   To calculate the average and standard deviation, we divide the FN and FP with window size (see the following equations (Eq: 6) and (Eq: 7)).

$$\frac{AvgFN}{WindowSize} * 100, \frac{StdFN}{WindowSize} * 100 \quad \text{(Eq: 6)}$$

$$\frac{AvgFP}{WindowSize} * 100, \frac{StdFP}{WindowSize} * 100 \quad \text{(Eq: 7)}$$

Chapter 5

Evaluation

5.1 Experimentation

In this chapter, we discuss all the experimentation details and how we evaluate the performance of our neural network models under real-world and synthetic data.

5.1.1 Implementation Details

Here, we explain the evaluation platform, the queries we implemented, the deep learning models we trained and their performance metrics. Basically, first we generated our dataset through operator execution. Secondly, we trained this data in our neural network.

1. **Evaluation Platform**

   We used 8 GB RAM and a 64 bit operating system. We run our code on a multi-core machine with 8 CPU cores and an Intel® processor 2.50 GHZ.

2. **Esper Framework**

   We used the .net framework version v4.6.1 and visual studio in 2017. We integrated nesper and nesper.IO library into a console application. The nesper library is a module for complex event processing (CEP) whereas the Nesper.IO library, input and output events. The significant benefit of using Nesper is that it can process a huge volume of incoming events. To read events from a file, we used CSV Input Output Adapter API and to write events in a file we used CSV Output Adapter API. The adapter adapts events from an external source and converts them into a format which can be processed by Esper [15]. To process events, EPService Provider provides administrator and runtime interfaces. The API administrator has two methods to process events; one is to create event patterns, and the other is to create EQL (event query language) statements.

   Create event patterns contain pattern operators like (every, timer: within, timer: interval, followed by, and, or) whereas EQL statements consist of select, from and where clause. EQl statements aggregate information from one or multiple event streams [15]. After setting up the framework, we modified the queries mentioned in chapter4.

3. **Esper Queries**

The first step in data generation was to apply esper queries on stock exchange and overtaking dataset. We modified the queries mentioned in section 4.1.1 and 4.1.2.

During the experiments, in case of a sequence and other operators, we set the the pattern size to 3, that means the operator has to match pattern(A B C).

(a) Query 1 consumes a pattern (A->B->C) of three distinct TradeEvents like A<>B<>C (operator "<>" means both events are not the same). When the pattern matches, every operator again look for another pattern. For example, event sequence is A1 B1 C1 B2 A2 D1 A3 B3 C3. The pattern to detect is A -> B -> C. Whenever B comes after A and C comes after B, the query matches the pattern. The same process recurs until the whole input stream is processed. The followed by -> operator ensures that first, the left side gets true and only then the right side can be examined for a pattern match.

```
-----------------------------------------------------------------
    Q1: Distinct Pattern (A B C)
-----------------------------------------------------------------
        engine.EPAdministrator.CreateEPL
        ("create window TenSecOfTicksWindow#time(10 sec) as
        TradeEvent");
        engine.EPAdministrator.CreateEPL
        ("on TradeEvent insert into TenSecOfTicksWindow select *");
        string epl = "SELECT * from pattern
        [every ((A=TradeEvent ->
        B=TradeEvent(A.symbol<>B.symbol) ->
        C=TradeEvent(B.symbol<> C.symbol)))]";
```

(b) Query 2, creates a pattern (A->B->C) of three distinct TradeEvents, and it also compares each events closing price with its opening price. Pattern matches when both events A and event B are detected.

```
 ----------------------------------------------------------------
     Q2: Detect Complex Pattern
-----------------------------------------------------------------
        engine.EPAdministrator.CreateEPL
        ("create window TenSecOfTicksWindow#length(10) as
        TradeEvent");
        engine.EPAdministrator.CreateEPL
        ("on TradeEvent insert into TenSecOfTicksWindow select *");
        string epl = "SELECT * from pattern
        [every (A=TradeEvent(A.close>A.open) ->
        B=TradeEvent(A.symbol<>B.symbol and B.close>B.open) ->
        C=TradeEvent(B.symbol<> C.symbol and C.close>C.open))]";
```

(c) Query 3, is listed in the MATCH-RECOGNIZE notation [33], which is concise and easy to understand. Complex events are detected between upper and lower limits when specific changes occurs in the stock prices. The lower limit and upper limit of the stock price is set. The query fetches out events which occur in between this range. Event A closing price should be higher than its lower limit, event B closing price should occur between lower and upper limits, and event C closing price should be higher than the upper limit. The Kleene* implies that one and many events can match. <,> are comparison operators.

```
-----------------------------------------------------------------------
     Q3: Stock pattern according to a range
-----------------------------------------------------------------------
        engine.EPAdministrator.CreateEPL
        ("create window TenSecOfTicksWindow#time(10 sec) as
        TradeEvent");
        engine.EPAdministrator.CreateEPL
        ("on TradeEvent insert into TenSecOfTicksWindow select *");
          var lowerlimit = 50;
          var upperlimit = 1000;
        string epl = "select * from TenSecOfTicksWindow
        match_recognize
        (partition by symbol measures A.ID as ID,A.symbol as symbol,
        A.close as close
        pattern(A B* C)
        define A as A.close >" + lowerlimit + ",
        B as B.close >" + lowerlimit + " and
        B.close <" + upperlimit + ",
        C as C.close >" + upperlimit + ")";
```

(d) Query 4, detect overtaking sequence (A->B->C->D). To detect overtaking sequence, Sequence(a1;b1;b2;a2) is followed. When an automobile "a" passes location L1, a window of size 10 is open. Another vehicle b appears in the same location L 1 and crosses the location L2 before vehicle "a" crosses L2. In this situation, "b" has overtaken "a" and thus violated the traffic rules.

```
-----------------------------------------------------------------------
    Q4: Detects an overtaking sequence
-----------------------------------------------------------------------
        engine.EPAdministrator.CreateEPL
        ("create window TenSecOfTicksWindow#time(1 min)as TradeEvent");
        engine.EPAdministrator.CreateEPL
        ("on TradeEvent insert into TenSecOfTicksWindow select *");
        string epl = "SELECT * from pattern
        [every (A=TrafficEvent(A.type='L1')->B=TrafficEvent
        (B.type='L1')-> C=TrafficEvent
        (C.number_plate=B.number_plate and C.type='L2') ->
        D=TrafficEvent(D.number_plate=A.number_plate and D.type='L2'))]";
```

4. **Data Preparation**

   The original dataset of the stock exchange and overtaking consists of all the unprocessed events. After the query filtered the complex events, these events are interpreted as processed.

   In the case of the stock exchange dataset, we executed the sequence operator to fetch the complex events. We obtained 851013 processed complex events and 438568 unprocessed events extracted from a dataset of 1289581 events.

   In the case of the vehicle overtaking dataset, we executed the sequence operator to detect overtaking complex events. We obtained 976 processed complex events and 9024 unprocessed events extracted from a dataset of 10000 events.

   The processed events are denoted as 0, and unprocessed events are represented as 1.

5. **Feed Forward Neural Network**

   We performed various number of experiments on stock exchange dataset using neural network. We designed the model according to the technical details mentioned in table 4.4, table 4.3 and table 4.5. We split our dataset into 80 percent training and 20 percent testing.

   First, we trained our model on single input and single output on stock exchange data. We train and test our model performance, and we got the following results as mentioned in table 5.1.

Table 5.1: Initial Experiments

Exp	Query	Title	Opt	Loss	Lr	Batch _Size	Epochs	Loss	Acc
1	Q1	Seq	adam	binary_ crossentropy	0.01	100	10	0.003	0.99
2	Q3	Range	adam	binary_ crossentropy	0.01	100	15	0.001	0.99
3	Q2	Stock Rise Seq	adam	binary_ crossentropy	0.01	100	20	0.69	0.51
4	Q1,Q2	Stock Rise Seq	adam	binary_ crossentropy	0.01	100	10	0.52	0.78

   Second, we trained our model on multiple inputs and multiple outputs according to the window size. In case of window size 10, we will have 10 events in each window where number of tuples in the data are considered as n number of windows. The first experiment of this architecture is performed on window size 10 that means we have 10 inputs and 10 outputs from our neural network model. Then we gradually increased the window size to 20, 30, 40 and 50. The below-mentioned table 5.2 and table 5.3 shows the train, test and sample size of sequence query for multiple size windows. The table 5.4 and table 5.5 displays the observed loss and accuracy after training the model. The target of the model is to predict whether the event should get a drop or not. The neural network model learned the patterns in the dataset and predicted the drop status of next incoming events.

Table 5.2: Stock Exchange, Query 1: Data Splits

Dataset	Query	Title	Window Size	Data Split	Test
Stock Exchange	Q1	Seq	10	Train on 82533 and Validate on 25792	1000
	Q1	Seq	20	Train on 41267 and Validate on 12896	1000
	Q1	Seq	30	Train on 27511 and Validate on 8598	1000
	Q1	Seq	40	Train on 20633 and Validate on 6448	1000
	Q1	Seq	50	Train on 16506 and Validate on 5159	1000

Table 5.3: Overtaking, Scenario, Query 4: Data Splits

Dataset	Query	Title	Window Size	Data Split	Test
Overtaking	Q4	Seq	10	Train on 640 and Validate on 200	100
	Q4	Seq	30	Train on 213 and Validate on 667	100
	Q4	Seq	50	Train on 128 and Validate on 40	100

Table 5.4: Stock Exchange, Query 1: Loss and Accuracy Observed

Exp	Query	Title	Window Size	Loss	Title	Opt	Lr	Batch	Epochs	Loss	Acc
5	Q1	Seq	10	binary_cross entropy	Seq	adam	0.01	100	100	0.50	0.70
6	Q1	Seq	20	binary_cross entropy	Seq	adam	0.01	100	100	0.51	0.72
7	Q1	Seq	30	binary_cross entropy	Seq	adam	0.01	100	100	0.56	0.70
8	Q1	Seq	40	binary_cross entropy	Seq	adam	0.01	100	100	0.56	0.70
9	Q1	Seq	50	binary_cross entropy	Seq	adam	0.01	100	100	0.57	0.70

Table 5.5: Overtaking, Query 4: Loss and Accuracy Observed

Exp	Query	Title	Window Size	Loss	Title	Opt	Lr	Batch	Epochs	Loss	Acc
10	Q4	Seq	10	binary_cross entropy	Seq	adam	0.01	100	100	0.29	0.90
11	Q4	Seq	30	binary_cross entropy	Seq	adam	0.01	100	100	0.26	0.90
12	Q4	Seq	50	binary_cross entropy	Seq	adam	0.01	100	100	0.26	0.90

5.2 Accuracy and Loss graphs

5.2.1 Stock Exchange, Query1:

The model accuracy and loss graph illustrates the data training and validation over a number of training epochs.

We observed from the loss plot that feed-forward neural network model has comparable performance on training and validation data. If these analogous plots start to drift consistently, then it indicates that training can be stopped. The loss shows how good or poor the model is performing on training and validation data after each iteration. Ideally, we observed in our graphs, the minimization in loss after a few epochs.

We observed from the accuracy plot that the feed-forward neural network model in some graphs probably can be more trained as the trend of accuracy in training and validation data is fluctuating over the last few epochs. Accuracy is calculated in the form of a percentage. It is an important metric to observe because it illustrates how accurate our model's prediction is as compared to the ground truth.

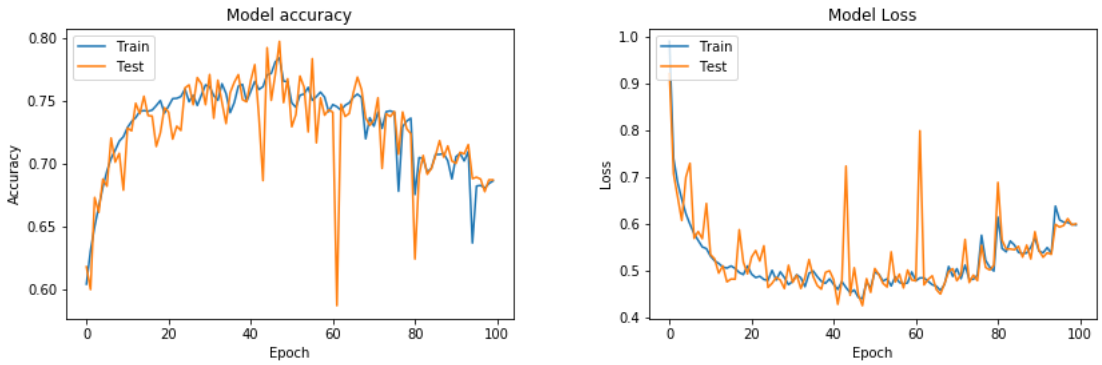

Figure 5.1: Accuracy and Loss for a window size 10 experiment

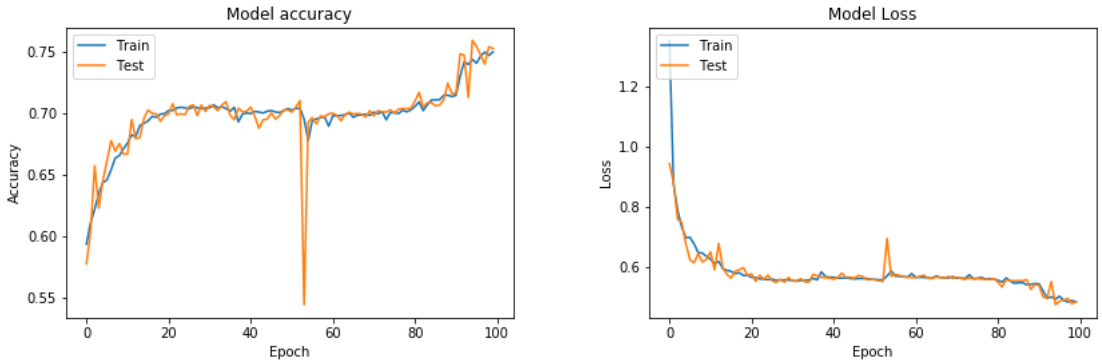

Figure 5.2: Accuracy and Loss for a window size 20 experiment

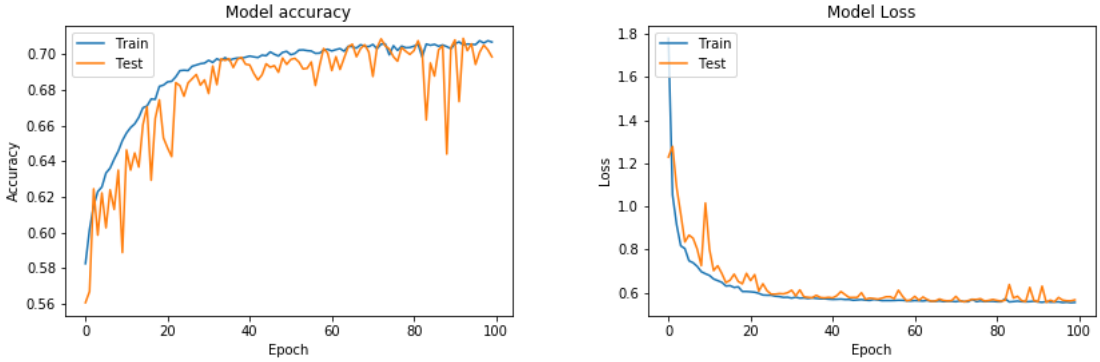

Figure 5.3: Accuracy and Loss for a window size 30 experiment

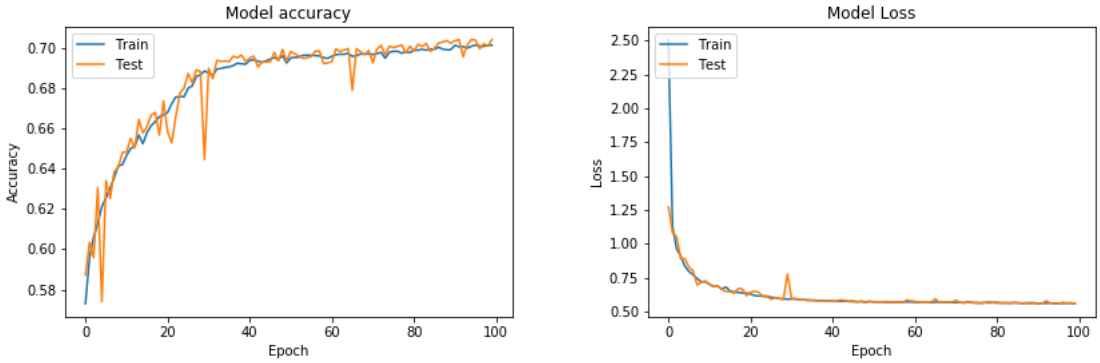

Figure 5.4: Accuracy and Loss for a window size 40 experiment

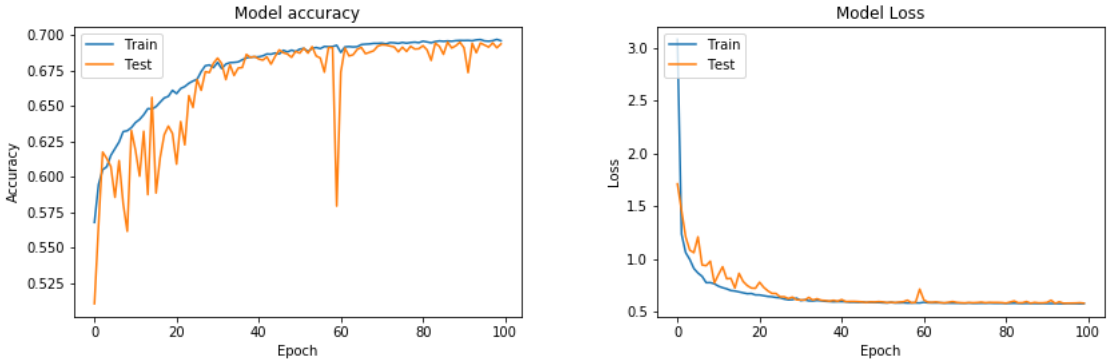

Figure 5.5: Accuracy and Loss for a window size 50 experiment

5.2.2 Overtaking Scenario, Query 4

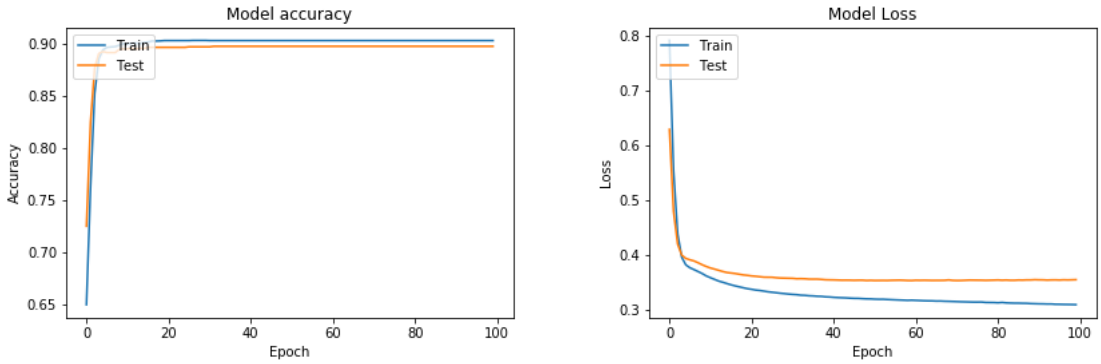

Figure 5.6: Accuracy and Loss for a window size 10 experiment

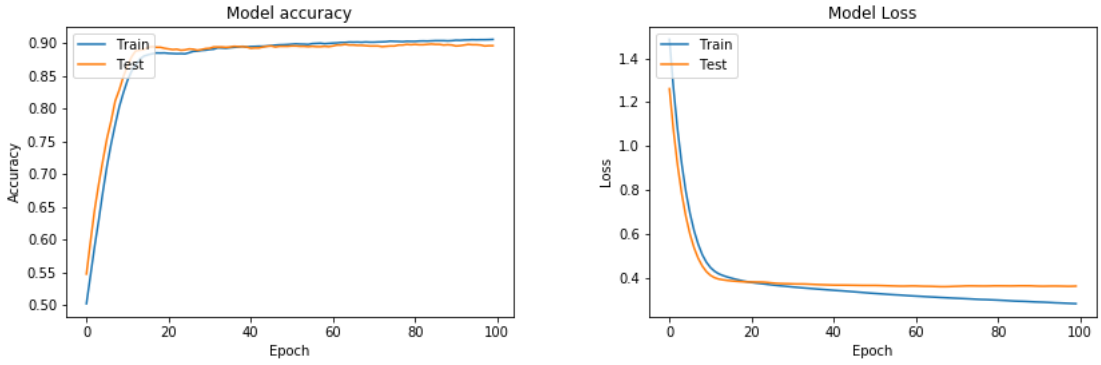

Figure 5.7: Accuracy and Loss for a window size 30 experiment

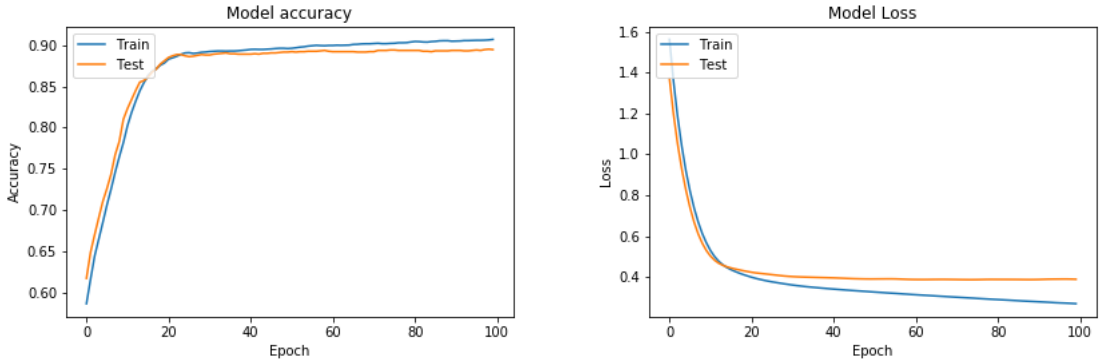

Figure 5.8: Accuracy and Loss for a window size 50 experiment

5.3 False Negatives and False Positives

Stock Exchange: Query 1

We plotted graphs of false negatives and positives for multiple window sizes ranging from 10 to 50 with maximum 25 percent drop in real and synthetic dataset.

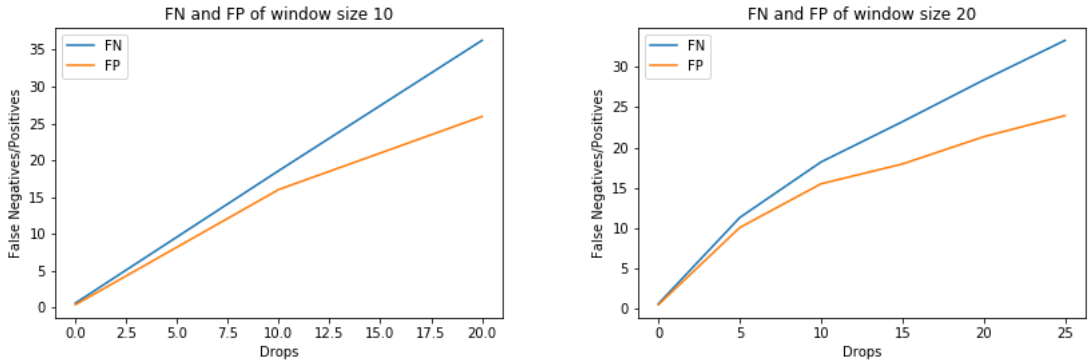

Figure 5.9: False negatives and positives for window size 10 and 20 on sequence operator

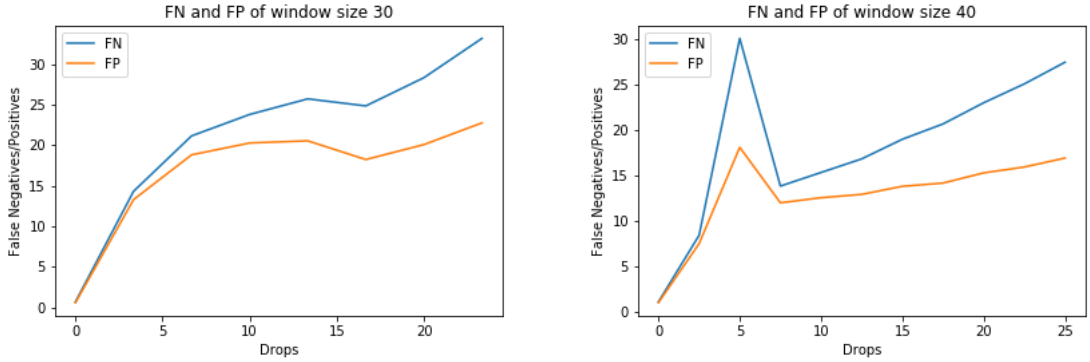

Figure 5.10: False negatives and positives for window size 30 and 40 on sequence operator

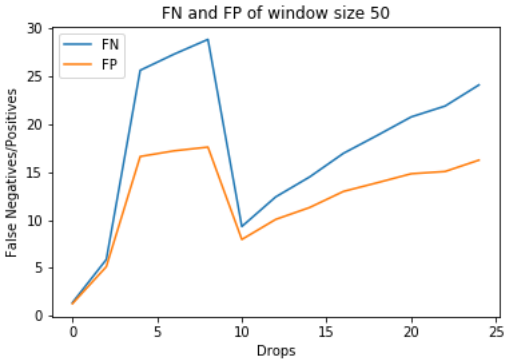

Figure 5.11: False negatives and positives for window size 50 on sequence operator

Overtaking: Query 4

The graphs show the number of false negatives and false positives in each window.

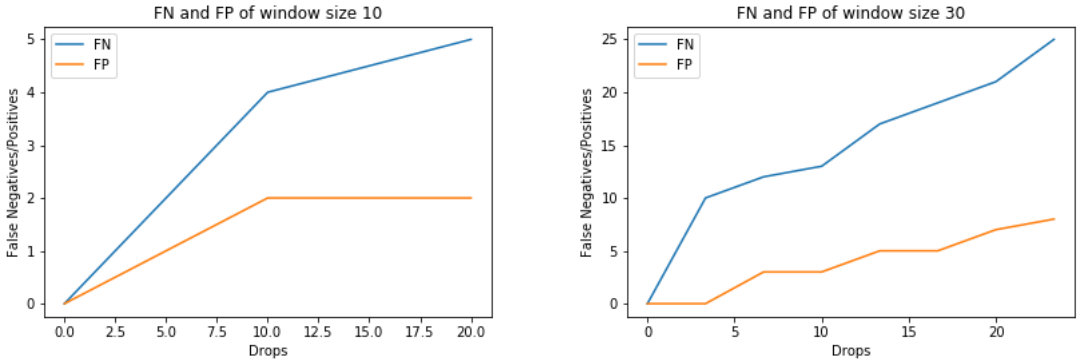

Figure 5.12: False negatives and positives for window size 10 on sequence operator

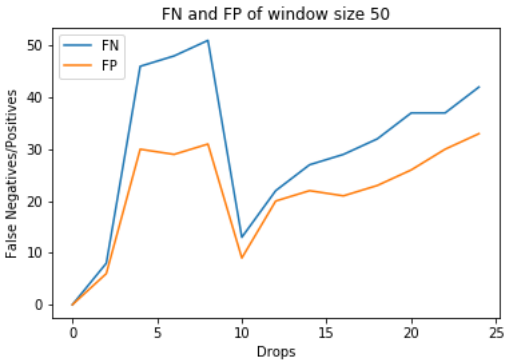

Figure 5.13: False negatives and positives for window size 50 on sequence operator

Conclusion:

It has been observed through experiments, higher drop rate leads towards higher False negatives and False positives. Some initial spikes are observed on synthetic and real data. However, with real-time data,in few experiments, a static growth in False negatives and False positives are observed as drop ratio increase. It is also observed with shorter window size, the FN and FP stay consistent as compare with drop ratio. With the suggested model we can estimate an ideal drop ratio against the given window sizes with maximum accuracy.

5.3.1 Standard Deviations

We plotted graphs of standard deviation for false negatives and positives for real and synthetic dataset both. Standard deviation measures the dispersion of data points from the average. A low standard deviation means, data points are close to the average. A high standard deviation means data points are more spread out.

Stock Exchange

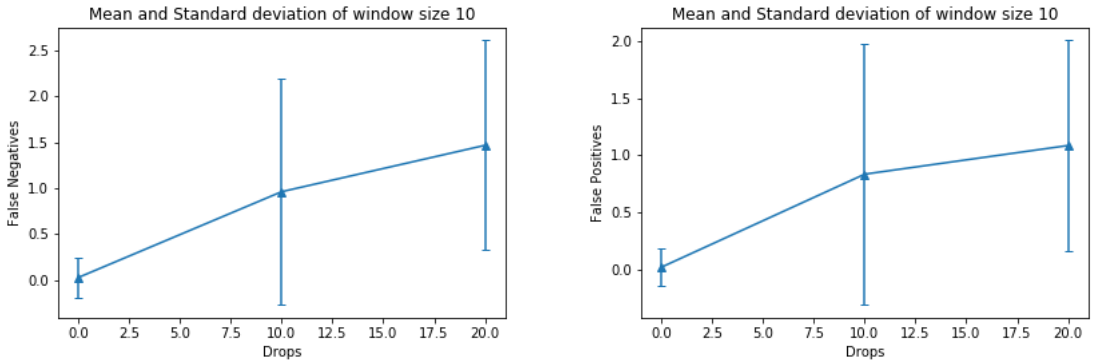

Figure 5.14: False negatives and positives for window size 10 on sequence operator

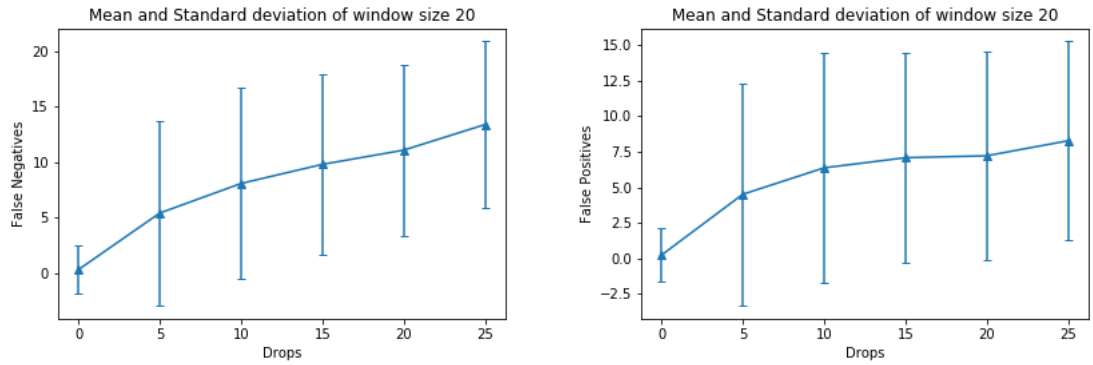

Figure 5.15: False negatives and positives for window size 20 on sequence operator

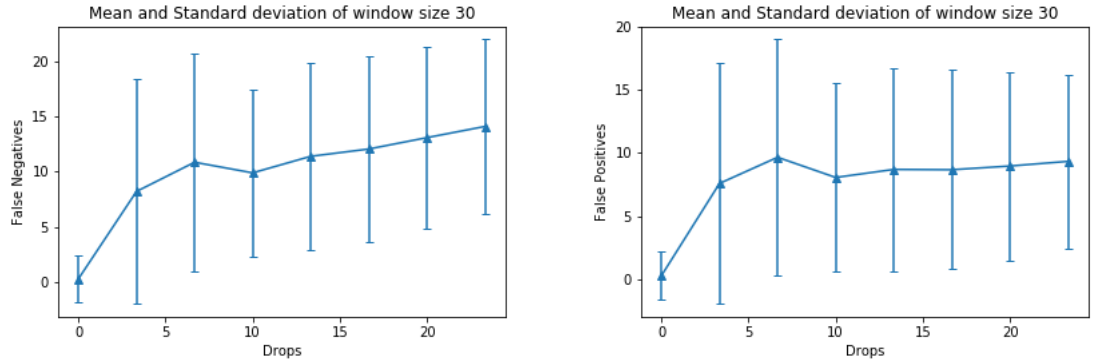

Figure 5.16: False negatives and positives for window size 30 on sequence operator

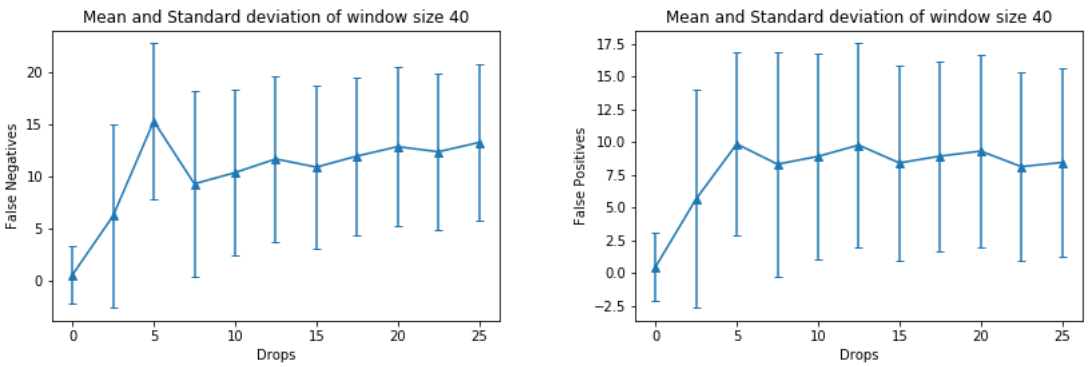

Figure 5.17: False negatives and positives for window size 40 on sequence operator

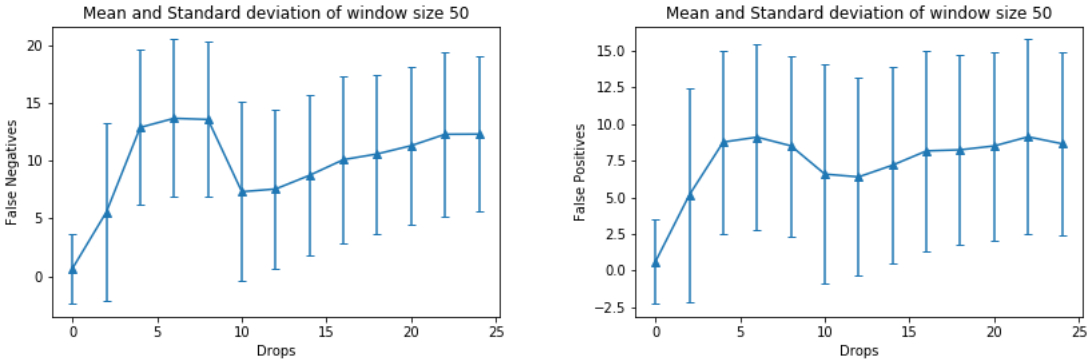

Figure 5.18: False negatives and positives for window size 50 on sequence operator

Overtaking

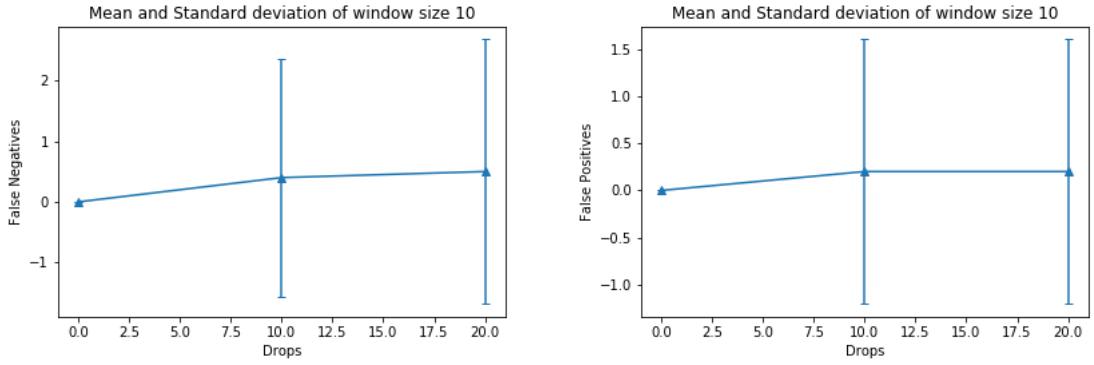

Figure 5.19: False negatives and positives for window size 10 on sequence operator

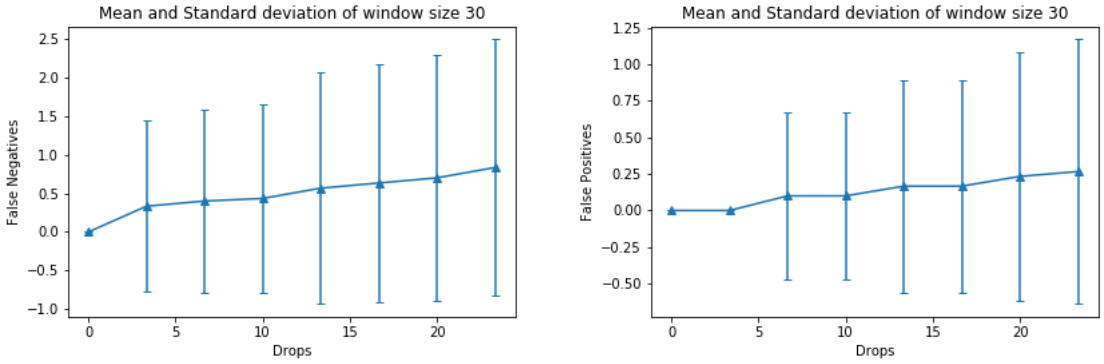

Figure 5.20: False negatives and positives for window size 30 on sequence operator

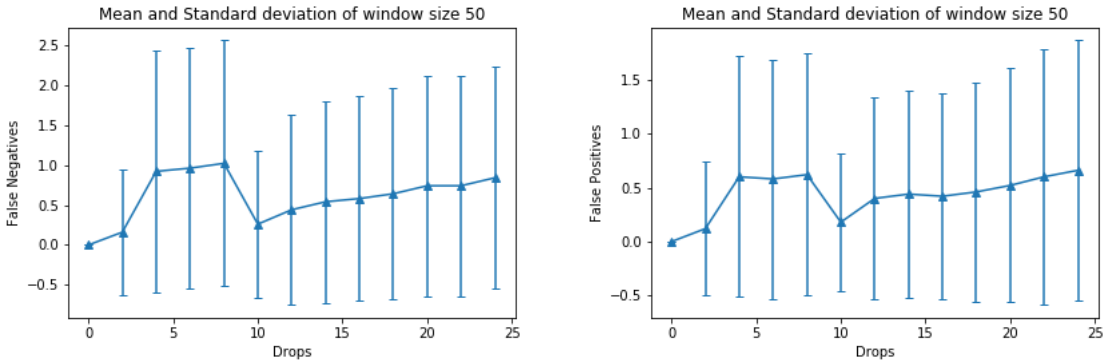

Figure 5.21: False negatives and positives for window size 50 on sequence operator

5.4 Comparison with state of the art

We implemented a generic black-box approach in which we only observed the incoming and outgoing events of the operator. Previously, In state of the art, the white box approach was implemented in which the researchers analyzed the internal working of the operator. These operators match user-specific patterns. If the operator is changed, then the white box is changed. The processing cost of simple events, complex events and a complete window is observed at each state of the operator. Researchers identified the quality of good and bad windows by carefully designing the pattern matching mechanism. The processing cost of each window is estimated at each state of the operator and the windows, which takes more processing cost were dropped. Markov reward cost process was used to learn the processing cost of windows, as a result model kept the good windows and dropped the bad ones. When distribution of event types changed in incoming event streams, model was unable to react accordingly. This approach has a correctness issue and is limited to a specific operator. This approach is not easily implementable in real-time.

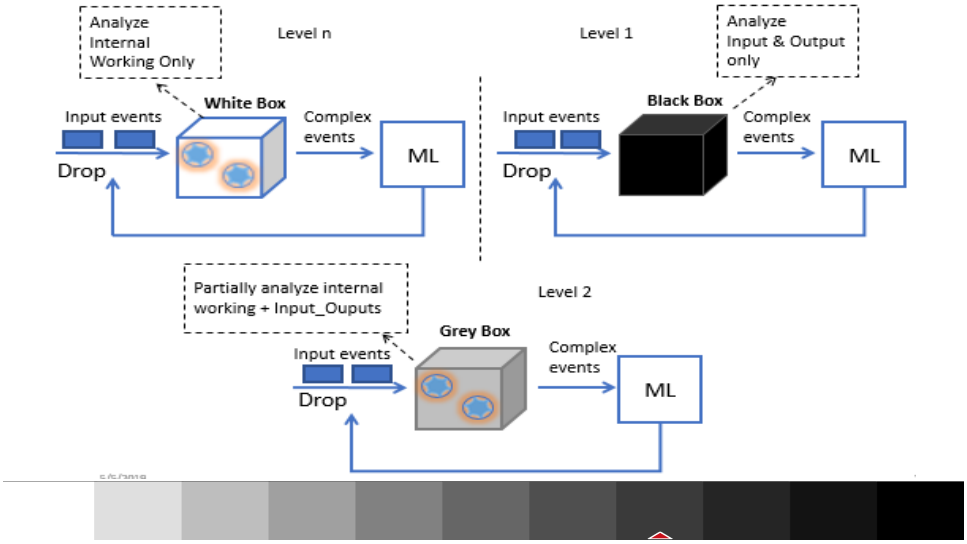

Figure 5.22: Future work: Spectrum

5.5 Conclusion and Future Work

In this research thesis we intelligently drop events. We trained the observational data on neural network and predicted the drop status on next incoming events. From the results, mentioned in section 5.3, it is concluded that with increase in drop rate there is a decrease in correctness. Correctness in this research refers to False -ves and +ves. The results indicated the percentage of False -ves and +ves at different drop rates. We selected 25 percent drop as an ideal drop rate.

In future work, we can further evaluate our model on different application scenarios with large window sizes. We can apply Deep learning models like LSTM to train the observational data. We can develop a grey box by partially observing the internal state of the operator. In the grey box, we can analyze the inputs, outputs along with, processing time, CPU consumption and memory utilization of complex events. We can gradually move from black to grey to white as illustrated in figure 5.22. and observe the correctness in results. We can do comparative analysis by evaluating correctness on whole spectrum if required between Black box and White box. It can be exciting to thrive in this area and see how far this thought can hold.

References

[1] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pp. 309–320, VLDB Endowment, 2003.

[2] Y. He, S. Barman, and J. F. Naughton, "On load shedding in complex event processing," *arXiv preprint arXiv:1312.4283*, 2013.

[3] F. A, "Window drop load shedding in complex event processing," *Universitätsstraße 38, 70569 Stuttgart, Germany*, 2018.

[4] A. Margara and G. Cugola, "Processing flows of information: From data stream to complex event processing," in *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, (New York, NY, USA), pp. 359–360, ACM, 2011.

[5] R. Mayer, A. Slo, M. A. Tariq, K. Rothermel, M. Gräber, and U. Ramachandran, "Spectre: Supporting consumption policies in window-based parallel complex event processing," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pp. 161–173, ACM, 2017.

[6] M. Kaiadi, "Artificial neural networks modelling for monitoring and performance analysis of a heat and power plant," 2006.

[7] C. Balkesen, N. Dindar, M. Wetter, and N. Tatbul, "Rip: Run-based intra-query parallelism for scalable complex event processing," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, (New York, NY, USA), pp. 3–14, ACM, 2013.

[8] R. Mayer, B. Koldehofe, and K. Rothermel, "Predictable low-latency event detection with parallel complex event processing," *IEEE Internet of Things Journal*, vol. 2, pp. 274–286, Aug 2015.

[9] T. De Matteis and G. Mencagli, "Parallel patterns for window-based stateful operators on data streams: an algorithmic skeleton approach," *International Journal of Parallel Programming*, vol. 45, no. 2, pp. 382–401, 2017.

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[11] R. Mayer, M. A. Tariq, and K. Rothermel, "Minimizing communication overhead in window-based parallel complex event processing," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, DEBS '17, (New York, NY, USA), pp. 54–65, ACM, 2017.

[12] B. Babcock, M. Datar, and R. Motwani, "Load shedding for aggregation queries over data streams," in *Proceedings. 20th International Conference on Data Engineering*, pp. 350–361, 2004.

[13] N. Rivetti, Y. Busnel, and L. Querzoni, "Load-aware shedding in stream processing systems," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pp. 61–68, ACM, 2016.

[14] T. d. Matteis and G. Mencagli, "Elastic scaling for distributed latency-sensitive data stream operators," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 61–68, March 2017.

[15] Esper, "Event Specification Language." http://www.espertech.com/esper/esper-documentation//, 2017.

[16] S. Singh and N. Singh, "Internet of things (iot): Security challenges, business opportunities reference architecture for e-commerce," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 1577–1581, Oct 2015.

[17] W. Fengjuan, Z. Xiaoming, W. Yongheng, and C. Kening, "The research on complex event processing method of internet of things," in *2013 Fifth International Conference on Measuring Technology and Mechatronics Automation*, pp. 1219–1222, Jan 2013.

[18] C. Balkesen, N. Tatbul, and M. T. Özsu, "Adaptive input admission and management for parallel stream processing," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 15–26, ACM, 2013.

[19] R. Mayer, A. Slo, M. A. Tariq, K. Rothermel, M. Gräber, and U. Ramachandran, "Spectre: Supporting consumption policies in window-based parallel complex event processing," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, Middleware '17, (New York, NY, USA), pp. 161–173, ACM, 2017.

[20] N. Tatbul, U. Çetintemel, and S. Zdonik, "Staying fit: Efficient load shedding techniques for distributed stream processing," in *Proceedings of the 33rd international conference on Very large data bases*, pp. 159–170, VLDB Endowment, 2007.

[21] B. Zhao, "Complex event processing under constrained resources by state-based load shedding," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 1699–1703, April 2018.