A COMPARATIVE STUDY ON STATISTICAL AND NEURAL APPROACHES FOR OPTIMIZING SUPPLY CHAIN MANAGEMENT (SCM) SYSTEMS

A PROJECT REPORT

Submitted by

HEEROK BANERJEE [Reg No: RA1511008010064]
DIKSHIKA K. ASOLIA [Reg No: RA1511008010214]
PRIYANSHI GARG [Reg No: RA1511008010224]
NIKHIL SHAW [Reg No: RA1511008010233]
GRISHMA SAPARIA [Reg No: RA1511008010251]

Under the Guidance of

Dr. V. GANAPATHY
(Professor, Department of Information Technology)

In partial fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

MAY 2019

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this project report titled "**A COMPARATIVE STUDY ON STATISTICAL AND NEURAL APPROACHES FOR OPTIMIZING SUPPLY CHAIN MANAGEMENT (SCM) SYSTEMS"** is the bonafide work of **"HEEROK BANERJEE [Reg No: RA1511008010064], DIKSHIKA K. ASOLIA [Reg No: RA1511008010214], PRIYANSHI GARG [Reg No: RA1511008010224], NIKHIL SHAW [Reg No: RA1511008010233], GRISHMA SAPARIA [Reg No: RA1511008010251]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. V. GANAPATHY
GUIDE
Professor
Dept. of Information Technology

Dr. G. VADIVU
HEAD OF THE DEPARTMENT
Dept. of Information Technology

Signature of Internal Examiner

Signature of External Examiner

ABSTRACT

The advent of AI tools in industrial management and business operations has broadly reinforced the interplay among business entities in the digital realm. These autonomous tools are undoubtedly powerful in employing self-learning and robust paradigms to facilitate predictive analytics for business intelligence but such tools still remain insufficient to overcome the impact of inevitable risks involved in businesses. In context to Supply Chain Management (SCM), it is therefore an open problem to eliminate and more significantly, optimize the impact of such risky operations on high-priority objectives such as Total cost, Lead-time and inventory costs that have been an evolutionary target for supply chain managers. In this study, we compare widely employed statistical and neural approaches to perform forecasting and conduct numerical analyses on multi-echelon supply chain networks. We start with experimental hypothesis testing on acquired datasets from multiple sources and hypothesize different batches of data to evaluate the measures of centrality and their co-relation attributes. This essentially reduces the input batches that are forwarded to forecasting models, hence relieving the system from computational overhead. We then proceed to evaluate machine learning models such as Decision Trees, Random Forests and Extended Gradient Boosting (XGB) Trees with pipe-lined architectures in order to observe their model and runtime performances on empirical and streaming data. We compare the obtained results to draw conclusions on their run-time performances. Next, we construct and train neural network models for use-cases such as function estimation and time-series analysis on forecasting problems related to risk-averse logistics and lead-time optimization. We have achieved outperforming results for Bi-directional Long Short-term Memory (LSTM) model and Non-linear Auto-Regressive (NAR) model for sequence-to-sequence prediction. We compare these models based on their performance, the loss function and their ability to generalize trends from the datasets. Finally, we conclude our study by commenting on the observed performances of the desired models and providing future directions to extend the contributions of this comparative study.

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to our guide, Dr. V. Ganapathy, (Professor Dept. of Information Technology) for his valuable guidance, consistent encouragement, personal caring, timely help and providing us with an excellent atmosphere for conducting research. All throughout the work, in spite of his busy schedule, he has extended cheerful and cordial support to our group for completing this research work. His suggestions and expertise played a key factor in redirecting our central focus to the major outcomes which are presented in this study.

We would also like to thank Dr. G. Vadivu (HoD, Dept. of Information Technology) for providing extra mural support and state-of-the-art infrastructure to us in order to facilitate this research work. We feel extremely privileged to acquire academic license for MATLAB™, JASP™and QtiPlot™software, without which this work could not have been completed.

We are also thankful to Dr. V. M. Shenbagaraman (HoD, Faculty of Management) for his insightful brainstorming discussions. We would like to thank Dr. D. Malathi (Professor, Dept. of Computer Science Engineering) for her anecdotal suggestions on performance tuning and simulation modelling in MATLAB. We also thank Dr. R. Subburaj for reviewing this work and for sharing his remarks.

Heerok Banerjee

Dikshika Asolia

Priyanshi Garg

Nikhil Shaw

Grishma Saparia

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SCM Supply Chain Management

SCRM Supply Chain Risk Management

RSCP Reverse Supply Chain Problem

DAG Directed Acyclic Graph.

RI Risk Index

LSTM Long Short-term Memory

NAR Non-linear Auto-Regressive

GA Genetic Algorithm

MILP Mixed-Integer Linear Programming

RSS Residual Sum of Squares

MSE Mean Squared Error

MLP Multi-Layered Perceptron

XGB Extended Gradient Boosting

IG Information Gain

LIST OF SYMBOLS

θ	Angle made by error surface to the plane of reference
$\sum_{i=1}^{n} A_i$	Summation of all elements in the set A from i=1 to n
$\prod_{i=1}^{n} B_i$	Product of all elements in the set B from i=1 to n

CHAPTER 1

INTRODUCTION TO SUPPLY CHAIN MANAGEMENT

SCM is the study of methods to run collaborative business operations in a way to maximize profit from the investments in the business pertaining to risk-averse strategies and optimizing resources, expenditures as well as effort to sustain a healthy growth. Nevertheless, it is clear that topological illustrations are used to represent business entities and their relationships. In network theory, Supply Chain Risk Management (SCRM) is understood as a Directed Acyclic Graph (DAG) (1). For example, shoppers, suppliers, makers and distributors are represented by vertices whereas the edges can be business activities like packaging, delivery, assembling, etc.

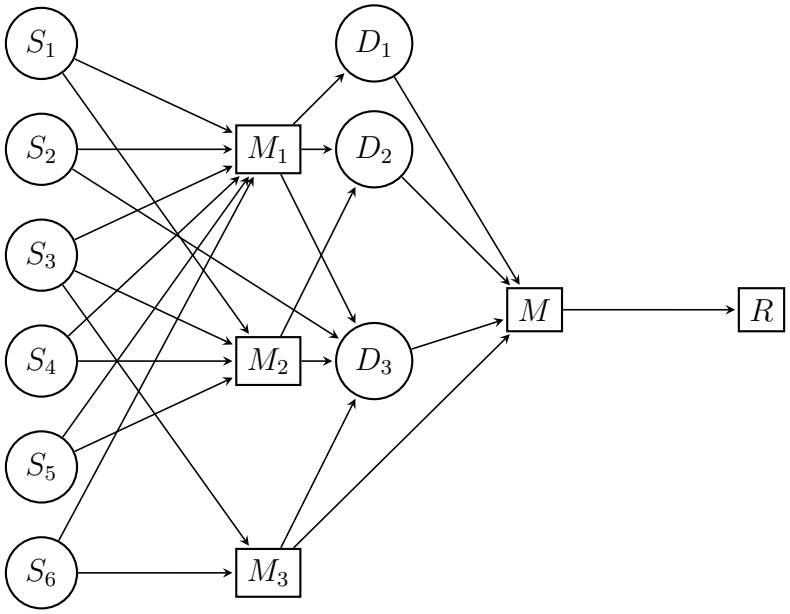


Figure 1.1: Supply chain as a complex network

In figure 1.1, the circular nodes denote an element in the supply chain network. The directed arrow denote a specific operation to be performed between two node. The complexity of the network is in-determinant of the number of nodes and the bi-lateral processes involved during a supply chain cycle (1). However, we have introduced the formulation of non-linear attributes such as risks in later chapters. As deducible from the inherent architectures of Supply chain networks, there are numerous issues attached with SCM relatable to social and economic factors (1).

1.1 Issues in Supply Chain Management

Psychological, social, cultural and personal factors influence consumer behaviour which is rapidly altered by globalization and technology. As social media users comply with new norms for interaction in the online virtual world, there is a need for the companies to use this new gigantic data to promote or make relevant products. The trends have a short cycle as the products which thrive and go extinct with them. Firms are under compulsion to keep producing new products as well as shipping new features while keeping the cost minimal. Furthermore, enhancing existing product features need revamping supply chain to aid product enhancement.

The emergence of social media has made the internet the biggest market where anyone can advertise and sell products to anyone across the earth. This has lifted the expectations of consumers for top standard products with innovation and consistency. Prominent trends like blockchain, big data, IoT, smart packaging are not only uplifting concurred standards but the methods in which they are administered and gauged. New advanced applications are required for handling, processing and making sense of gigantic data being generated. The immediate issue an enterprise face is whether investing in utilizing platforms established on micro-services and big data will bear data lifting essentials.

Hence, it is clear that the overgrowth of democratized technologies across the globe is a harbinger for severe competence in the contemporary market. The impact of these inevitable components are only some of the attributes which surround the problems related to SCM. In this research study, we overlook the quantitative factors involved in business operations and provide a context to transform operation inclined toward fault-tolerant and risk-aware decision making. These essentially brings new trade-offs to managers but it is necessary to outline the tools applicable for dealing with multi-objective decision making. So, the central focus of our study is on reversing effects of risks and understand the non-linearity of this quantity from an optimization approach. However, the goal of this study is to provide substantial contributions to modelling risks are non-dominant and provide a model-to-model comparison on quantitative measures for evaluating a risk-aware logistics scenario in a business setting.

.

1.2 Risks involved in Supply Chains

1. **Supply risk**- It is the pervasive factor that determines the extent of variations in supply trends and supply chain outcomes. Broadly speaking, it can be defined as a conflict between supply and demand which can disrupt consumer life and welfare. Supply risks occurs due to affairs associated with individual insufficiency and severe competence (4).

2. **Demand risk** - A risk of encountering unusual consumer demand is inevitable due to socio-economic changes. A high forecast but low actual demand bears additional costs for a business firm in terms of discarding or depositing their abundant assets (4). Conversely, low forecast but high actual demand supplements opportunity costs in terms of bygone sales. It is routinely remarked in marketing, sales, capital investments and supply chain decisions to be a construct of demand forecasts (5).

3. **Operational risk**- Operational risk is the cumulative factor that describes the probability of enduring loss due to a potential failure of a business operation (5). Operational risks results from unprecedented failure of operations (5); regressive manufacturing endeavours and incapacitated processing, high degree of deviation and advent of incoming new trends and technology(5).

4. **Security risk**-Something or someone likely to cause danger or difficulty.Security risks arrives due to infrastructural security, systems security, vandalism, villainy (7). A breach in technical infrastructure or a conflict of interests are also accounted as security risks as they promote an extent of unnatural behaviour.

5. **Social risk**-Social risks are described as the arrival of challenges that business practices endure, due to impacts propagated through the social participants of the business hierarchy. (3).

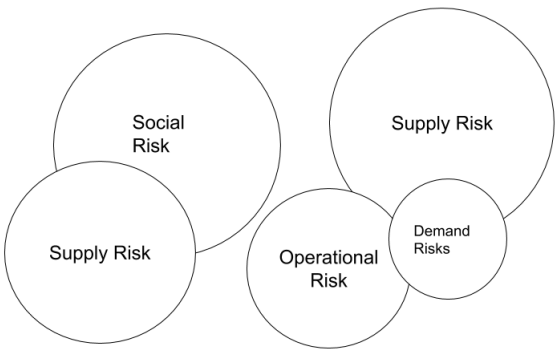


Figure 1.2: Venn Diagram for representation of risks involved in supply chain systems

CHAPTER 2

REVIEW OF LITERATURE

Optimization problems in supply chain systems have attracted researchers and supply chain managers since the early industrialization and democratization of products at the expense of outsourcing services like manufacturing, dispensing, transportation, packaging etc. Optimum resource configuration is one of the most attributed problems in SCMS (12; 13). For supply chain systems that are immune to pervasive risks, supply chain managers emphasis on determining singleton optimization problems such as total costs, Return on Investment, transportation cost etc. However, framing single objective-based supply chain needs prior exposure and additional supportive functions such as lead-time optimization, stock optimization, inventory-level optimization etc (14). Sometimes conflicting objectives can cause disruptions in the supply chain; and therefore, a thorough study outlining the trade-offs between objective functions is necessary prior to mathematical modelling. This deviates the attention from supply chain modelling towards satisfying conflicting objectives (15).

2.1 Multi-objective Optimization

Many novel supply chain evaluation models have been proposed with limited constraints and succinct objectives. For example, A location-inventory problem was investigated using a multi-objective self-learning algorithm based on non-dominated sorting genetic algorithm II (NSGA II) in order to minimize the total cost of the supply chain and maximize the volume fill rate and the responsiveness level (16). In another study, a multi-objective programming model using Fuzzy logic was derived for forward and reverse supply chain systems for minimizing total cost and the environmental impact. The Fuzzy-based model consisted of two phases. The first phase converts the probability distribution of multi-objective scenarios into mixed integer linear programming model and then into an auxiliary crisp model (17). The second phase generates fuzzy decomposition methodology based on the subjected linear inequalities and variable constraints

to search for the non-dominant solution (18). Several models have also been proposed for multi-echelon supply chains with closed-loops. For example, A multi-product with multi-stage and multi-period random distribution model is proposed to compute with predetermined goals for a multi-echelon supply chain network with high fluctuations in market demands and product prices. Similarly, a two-phased multi-criteria fuzzy-based decision approach is proposed where the central focus is to maximize the participant's expected revenues, average inventory levels along with providing robustness of selected objectives under demand uncertainties (20). Considering travelling distance and traveling time, another multi-objective self-learning algorithm is proposed namely fuzzy logic non-dominated sorting genetic algorithm II (FL-NSGA II), which solves a multi-objective Mixed-Integer Linear Programming (MILP) problem of allocating vehicles with multiple depots based on average traffic counts per unit time, average duration of parking and a primitive cost function (20). Alternatively, a bi-objective mathematical model is formulated which searches for local minimas of total cost and transportation cost; which attempt a pareto-search optimal checks to plot the pareto optimal fronts. A new hybrid methodology is also proposed which combines robust optimization, queuing theory and fuzzy inferential logic for multi-objective programming models based on hierarchical if-then rules (22). Another variant of the above soft-computing approaches consisting of a swarm-based optimization technique namely "The Bees algorithm" is discussed and implemented to deal with multi-objective supply chain problems; to find pareto optimal configurations of a given SCM network and attempts to minimize the total cost and lead-time. For an optimization model integrating cost and time criteria, a modified particle-swarm optimization method (MEDPSO) is adopted for generating solutions of a multi-echelon unbalanced supplier selection problem (24; 25; 26).

2.2 Bayesian Modelling for Risk Analysis

Bayesian logic is one of the most fundamental method borrowed today for optimizing and reinforcing probabilistic models (10). Quantitative measures that are found dispersed in any objective based problem are often modelled using Bayesian analysis (12). For example, Supply chain network as Directed Acyclic Graph (DAG) welcomes the expansion of conditional probability and measures of similarity to examine the extent

of how such graphs can be extrapolated and utilized to infer vague quantities. In contrast to fuzzy logic, where no deterministic model is presumably evaluated, Bayesian modelling assumes the underlying principals of probability distributions to derive conclusions (9). Such methods are extraordinarily employed in modelling risks in supply chain networks. Capturing the effects of risk propagation and assess the total fragility of a supply chain are considered one of the most significant problems in SCRM. The amount of uncertainty and inevitability in supply chain operations transforms this problem to multi-criteria based analytical problems. However, several models were summarized and reinforced to create a well-acknowledged and succinctly derived model for formulating disruption probability and the occurrence of risks. Based on Inventory optimization literature, it is evident that supply risk originates in multiple locations and propagate autonomously within the network. The effect is identical to and termed as "Ripple effect".

2.3 Reverse Supply Chain Problems

As per the investigation conducted within a limited scope, design strategies and managerial architectures for reverse supply chain systems are unexplored and lacking confirmed literature (8). Optimization problems pertaining to lead-time optimization, inventory level optimization and cost optimization are traditionally dealt with G (GA) based approaches by spawning a subset of ideal solutions and evaluating fitness values for each solution (18; 19; 22). Heuristic algorithms tend to search for local optimal values and have better dropout ratios as compared to GA-based approaches. On the other hand, evolutionary schemes tend to combine optimization methods with GA approaches (8). For example, a customer-allocation optimization is attempted using MILP based on multi-tier retailer location model. The model examined the locations and capacities of manufacturing cum retailing facilities as well as transportation links to better facilitate customers with criteria-based requirements. In (8), an Reverse Supply Chain Problem (RSCP) model for addressing the design of reverse manufacturing supply network is proposed. The model is a hybrid model consisting of GA-based approaches and underlying 0-1 MILP optimization constraints.

CHAPTER 3

RISK MODELLING IN SUPPLY CHAIN NETWORKS

In this chapter, a deterministic mathematical model for each attribute in the supply chain network is discussed. We have used MILP for formulating risk indices.

3.1 Mixed Integer Linear Programming (MILP) Model

For a given supply chain with 4 echelons: Supplier, Distributor, Manufacturer, Retailer and selective activities such as sourcing or supplying raw material to each component, assembling final products and delivering to destination markets, each component has its own cost, lead-time and associated risk.

1. The Risk Index (RI) is derived from the model proposed in (9), and can be mathematically formulated as:

$$RI_{supplier} = \sum_{i=1}^{n} \alpha s_{ij} \,.\, \beta s_{ij} \,.\, (1 - (1 - \prod_{j=1}^{m} P(\tilde{S}_{ij})) \tag{3.1}$$

where, αs_{ij} is the consequence to the supply chain if the i^{th} supplier fails,
βs_{ij} is the percentage of value added to the product by the i^{th} supplier,
$P(\tilde{S}_{ij})$ denotes the marginal probability that the i^{th} supplier fails for j^{th} demand,

Similarly, the risk indices for the rest of the components can be calculated as:

$$RI_{distributor} = \alpha d_{risk_i} \,.\, \beta m_i \,.\, (1 - (1 - P(\tilde{M}_j))) \tag{3.2}$$

$$RI_{manufacturer} = \alpha m_{risk_i} \,.\, \beta m_i \,.\, (1 - (1 - P(\tilde{M}_j))) \tag{3.3}$$

$$RI_{retailer} = \alpha r_{risk_i} \,.\, \beta r_i \,.\, (1 - (1 - P(\tilde{R}_j))) \tag{3.4}$$

2. For each set of demand, the cumulative risk index of the supply chain network can be calculated as:

$$TRI = w_1 \,.\, RI_{supplier} + w_2 \,.\, RI_{distributor} + w_3 \,.\, RI_{manufacturer} + w_4 \,.\, RI_{retailer} \tag{3.5}$$

where, w_1, w_2, w_3, w_4 are arbitrary weights such that $w_1 + w_2 + w_3 + w_4 = 1$

3. The total supply chain cost can be calculated as (**?**):

$$TC = \xi \times \sum_{i=1}^{N} (\mu_i \, . \, \sum_{j=1}^{N_i} C_{ij} y_{ij}) \tag{3.6}$$

where, N is the number of components,
ξ denotes the period of interest,
μ_i is the average demand per unit time,
C_{ij} is the cost of the j^{th} resource option for the i^{th} component,
y_{ij} is a variable denoting the amount that the i^{th} component supplies as a participant for the j^{th} resource option

4. For each couple of normalized risk index and total cost of the supply chain, the main objective function Z is given as:

$$Z = w_1 \, . \, TC_n \; + \; w_2 \, . \, TRI_n \tag{3.7}$$

where, TC_n is the normalized total cost,
TRI_n is the normalized total risk index,
w_1, w_2 are the weights ; $w_1 + w_2 = 1$

3.2 Numerical Analysis

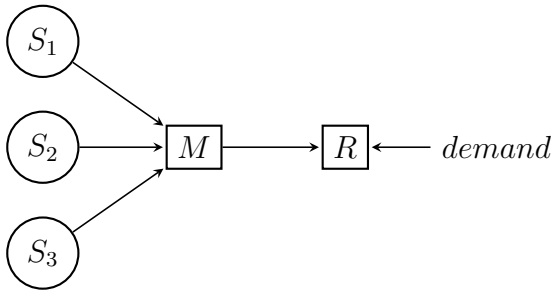


Figure 3.1: Simple Supply chain architecture with multiple suppliers, single manufacturer and single retailer

Consider a simple supply chain network as a directed graph G comprising of suppliers $S = \{S_1, S_2, S_3\}$, a manufacturer M and D, a retailer R as shown in Fig.3.1. For a realistic supply chain scenario, one must assume that the supply chain actors are independent and consequently the logistics and operational costs can be arbitrarily chosen. Table 1 represents the configuration of the pseudo supply chain model given in Fig.3.1 :

Table 3.1: Example of supply chain logistics

Component Name	Stage Cost	Stage time	Avg Demand	StdDev
S_1	\$2	8	25	2
S_2	\$3	15	50	7
S_3	\$7	32	120	14
M	\$0	10	300	30
R	\$4	10	300	25

In the above architecture, the subjected demand to the manufacturer is propagated directly to the manufacturer. Consequently, the demand is arbitrarily distributed among the suppliers $\{S_1, S_2.S_3\}$. For example, if the demand is 'd' and the demand distribution is $\{x_1, x_2, x_3\}$, then the cost function F_c can be represented as a linear Diophantine equation:

$$F_c(G, S) : 2x_1 + 3x_2 + 7x_3 \leq 4d \tag{3.8}$$

$$x_1 + x_2 + x_3 = d \tag{3.9}$$

For example, let us consider that the demand subjected to the supply chain model is 100 units. In that case we form a system of two linear equations, from which we can obtain a set of possible solutions. For a simplistic case study, we have assumed that the linear inequality is a linear equality subject to linear constraints, given that the supply chain model is at a supply-demand equilibrium. This implies that supplementary costs outside the scope of the suppliers are neglected and the overall expenditure is equated to the total revenue generated in one echelon.

$$2x_1 + 3x_2 + 7x_3 = 400 \tag{3.10}$$

$$x_1 + x_2 + x_3 = 100 \tag{3.11}$$

Multiplying equ.(3.11) by 3 and subtracting from equ. (3.10),

$$4x_3 - x_1 = 100 \tag{3.12}$$

$$x_3 = \frac{100 + x_1}{4} \tag{3.13}$$

Assume $x_1 = a$, then

$$x_3 = \frac{100 + a}{4} \tag{3.14}$$

clearly, a must a multiple of 4 since x_3 should be a positive integer.

Substituting x_3 and x_1 in terms of a, we get

$$a + \frac{100 + a}{4} + x2 = 100 \tag{3.15}$$

$$x2 = 100 - a - \frac{100 + a}{4} \tag{3.16}$$

Table 3.2: Non-negative integral solutions

Supplier 1 (S_1)	Supplier 2 (S_2)	Supplier 3 (S_3)	Cost Function
0	25	75	600
4	26	70	576
8	27	65	552
12	28	60	528
16	29	55	504
20	30	50	480
24	31	45	456
28	32	40	432
32	33	35	408
36	34	30	384
40	35	25	360
44	36	20	336
48	37	15	312
52	38	10	288
56	39	5	264
60	40	0	240

The linear Diophantine equation can be solved employing the SymPy library in Python. A psuedo code is given below:

```
from sympy.solvers.diophantine import diophantine
from sympy.solvers.diophantine import diop_linear
```

```
from sympy import symbols,sympify

a,b,c=symbols("x,y,z", integer=True)


coeff=list()
for key in suppliers:
    #print(key)
    coeff.append(suppliers[key])
#print(coeff)

### Base Solution
demand=500
man_cost=38

base_expr=2*a+3*b+7*c-4*demand
print(base_expr)
[base_sols]=diophantine(base_expr)
print("Base Soln:"+ str(base_sols))
t_0,t_1=base_sols[2].free_symbols
```

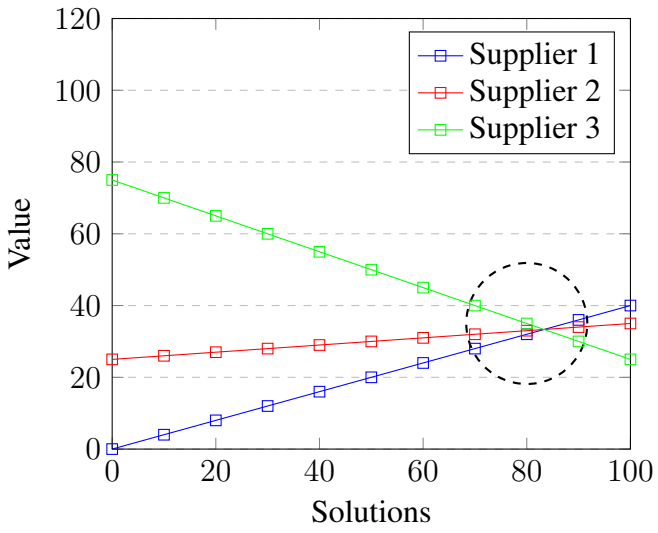


Figure 3.2: Non-negative integral solutions of Diophantine equation with d=100

The monotonous change in supplier distribution as the solutions increases over x axis suggests that the minimum likelihood for an appropriate distribution is only concentrated where the demand is equally subject to the supplier. The encircled region in Fig. 3.2 denotes the minimum likelihood for risk-propagation.

CHAPTER 4

STATISTICAL APPROACHES

In this chapter, we will introduce descriptive hypothesis testing and statistical machine learning models for problems related to demand forecasting and curve-fitting. In order to observe the nature of the models, We will test the selected models with large datasets acquired from multiple sources. Consequently, we will tabulate the achieved evaluation metrics and attempt to tune the model parameters in order to optimize the limitations and boost their individual performances. Finally, we will illustrate a well-structured nomenclature to present a generalized conclusion drawn from the observed scenarios.

4.1 Hypothesis Testing

4.1.1 Student's T Test

Descriptive hypothesis testing, sometimes referred as confirmatory data analysis is a method of statistical inference on the basis of observing events and statistical measures such as measures of central tendency and measures of shape of distribution. A statistical hypothesis is a hypothetical statement about a sample or a population of samples, which describes the relationship between two distinct variables and both the variables lead to specific effects on the other. In order to validate a statistical hypothesis, we conducted a series of parametric tests such as student's T test. A student's T test is required to compare the measures of central tendency of two samples. Essentially, one sample is taken from the original population and their respective means are calculated.

First we hypothesize a statement as follows:

$$H_0\ (null\ \ hypothesis)\ : \mu_1 - \mu_2 \leq 0; \tag{4.1}$$

And, another statement such as:

$$H_1\ (alternative\ \ hypothesis)\ : \mu_1 - \mu_2 > 0; \tag{4.2}$$

Table 4.1: Student's T test for "UAE Distributor" dataset: Column E (Sales)

Sample Size (%)	Population Mean	Sample Mean	Hypothesis Parameters t	p	Decision
20	391.27	409.97	-1.95	0.97	Accept
25		404.91	-1.56	0.94	Accept
30		408.18	-2.07	0.98	Accept
33		406.83	-1.97	0.97	Accept
38		404.64	-1.79	0.96	Accept
40		404.99	-1.86	0.96	Accept
48		401.79	-1.53	0.93	Accept
50		399.36	-1.19	0.88	Accept
70		395.59	-0.71	0.76	Accept
99		391.57	-0.05	0.52	Accept

[a] There is no significant difference in the population mean and sample mean across variable sample sizes.
[b] A conclusion can be drawn that the dataset is uniformly distributed across all 6011 samples and a sample can be extracted and treated as a representation of the entire population.

Table 4.2: Student's T test for "UAE Distributor" dataset: Column F (Cost)

Sample Size (%)	Population Mean	Sample Mean	Hypothesis Parameters t	p	Decision
20	726.70	743.19	-0.44	0.67	Accept
30		781.56	-1.70	0.95	Accept
35		771.65	-1.49	0.93	Accept
40		755.87	-1.02	0.84	Accept
45		752.71	-0.96	0.83	Accept
50		742.63	-0.61	0.72	Accept
60		737.74	-0.45	0.67	Accept
65		734.38	-0.32	0.62	Accept
70		727.92	-0.05	0.52	Accept
80		752.71	-0.96	0.83	Accept

[a] There is no significant difference in the population mean and sample mean across variable sample sizes.

The two statistical tests conducted on the same dataset concludes that a percentage of sample extracted from the population can be assumed to be an appropriate representative of the population and later utilized for descriptive analytical tasks such as input-output curve fitting. This promises to reduce redundancy and computational complexity.

Table 4.3: Student's T test for "Walmart" dataset : Column K (Weight KG)

Sample Size (%)	Population Mean	Sample Mean	Hypothesis Parameters t	Hypothesis Parameters p	Decision
30	48,818.57	10,405.42	0.69	0.24	Reject
15		12,405.21	0.46	0.32	Reject
44		9,447.93	0.86	0.19	Reject
65		9,243.03	1.05	0.14	Reject
72		8,860.44	1.11	0.13	Reject
82		10,320.07	1.14	0.12	Reject
10		11,025.16	0.39	0.34	Reject
7		7,759.20	0.35	0.36	Reject

[a] The null hypothesis is rejected.
[b] The data is heavily skewed and assuming that the distribution is normal, we observe that the difference in means is significantly greater for most of the sample sizes, hence we cannot arbitrarily choose any of these sample sizes for our purpose.

4.1.2 Mean Normalization

Mean Normalization is a technique of reducing the marginal variance in a set of data points by representing a series by its measures of central tendency such as mean. Mean normalization helps in diluting errors in the data by interpolating statistical representation of data points from which the original set can be derived. Mathematically, the mean normalization is performed as follows:

$$\chi = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.3}$$

$$\hat{x_i} = |x_i - \chi| \tag{4.4}$$

During the course of this study, we reviewed many datasets obtained from multiple sources. Generally, empirical datasets extracted from physical experimental setup are more likely to contain marginal errors, instrumentation error, human error etc. and therefore it is crucial to eliminate the redundant error percentage indigenous in such datasets. We have transformed the obtained datasets into normalized datasets and redesigned some of the machine learning models as well as ANN models to observe their learning behaviour from such tightly encapsulated data. The normalized data series

however retains the original information provided that the normalization process is not stringent. We must carefully observe the standard deviation of the original series and reasonably select a batch size. Consequently, it is concluded within the scope of this study that the error percentage is drastically reduced when the samples are prepossessed with appropriate measures.

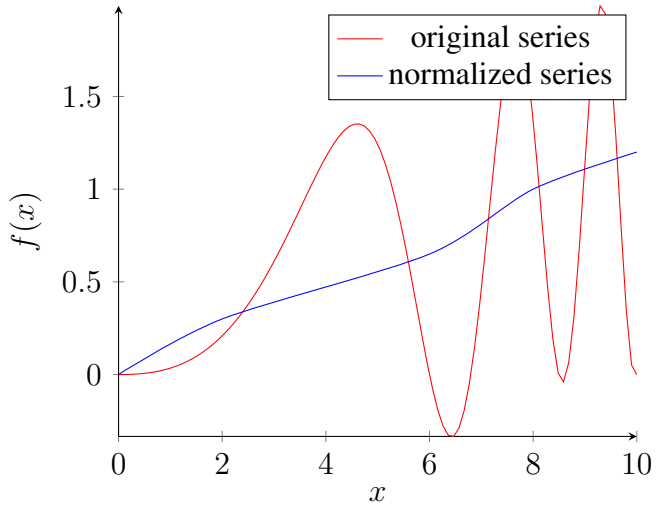
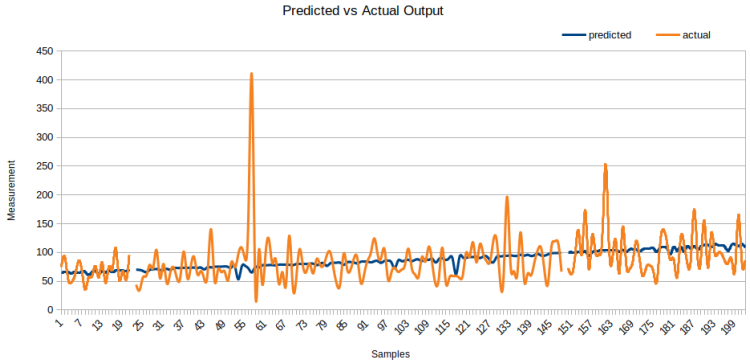


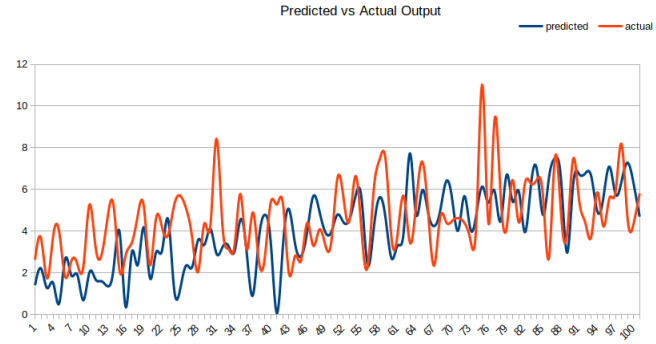
Figure 4.1: Representation of original data vs Normalized data series

The pseudo code given below illustrates the algorithmic steps of normalizing a dataset using MATLAB ™. As per the pseudo code, samples are grouped in order of 100s and their respective means are computed. The mean is substituted as an informative datapoint to yield a axiomatically similar dataset as the original one. This essentially decomposes the dataset by reducing the number of samples to be processed but also preserves the integrity of the data. Hence, it is a fruitful approach when considering high volume of noisy data along with huge sample size.

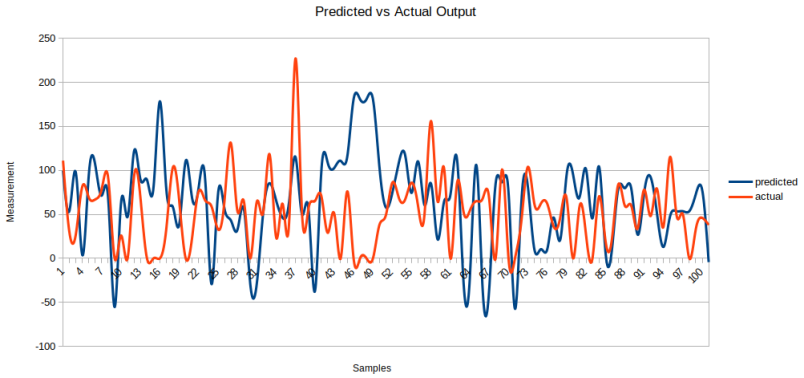
```
% Batched Mean Normalization- Batch_size=100
sample_size=170000;
batch_size=100;
batch_input100=[];
for i=0:(sample_size/batch_size)
    temp=data_input(1+i*batch_size:batch_size*(i+1),1);
    batch_input100=[batch_input100; mean(temp)]
end
```



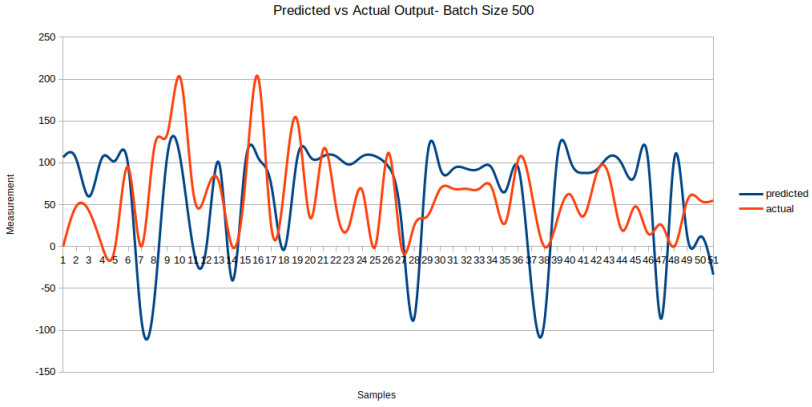
(a) Over-generalization



(b) Normalization: Batch_size=100



(c) Normalization: Batch_size=200



(d) Normalization: Batch_size=500

Figure 4.2: Actual vs Predicted plot for Normalized data with different batch sizes

4.2 Machine Learning Models

A Machine Learning model is a composite processing system that attempts to perform statistical inference based on input-output patterns. Statistics and Machine Learning Toolbox provides functions and apps to describe, analyze, and model data. You can use descriptive statistics and plots for exploratory data analysis, fitting probability distributions to data, generating random numbers for Monte Carlo simulations, and performing hypothesis testing.

There are broadly two categories of problems in Machine Learning namely regression and classification problems. A regression problem is defined as an estimation problem where the target variable is a continuous variable whereas in a classification problem, the target variable is categorical. For the sake of a comparative study, we will discuss each model extensively and compare the results in section 5.3. Regression and classification algorithms allows you to draw inferences from data and build predictive models.

4.2.1 Regression Model

For a regression problem, we have considered the dataset "UAE_Distributor.xlsx"[see Appendix A.2] and selected 2 features "Net Sales" and "Net Cost" as input features and "Average Sales Quantity" as the target feature . Hence the linear regression problem can be formulated as:

$$y = \theta_1 \, x_1 + \theta_2 \, x_2 \tag{4.5}$$

where, y denotes the average sales quantity,
x_1 denotes the net sales and s_2 denotes the net cost

4.2.2 Decision Tree Model

Decision Trees are predictors that interprets decisions based on path traversals in a structured tree beginning from the root node to a leaf node. The leaf node essentially remarks a decision. Every parent node in the decision tree denotes an arbitrary binary test that concludes in either 'True' or 'False' for a classification problem, or results in a real integer. Regression trees measure these real-valued integers to give numeric responses. The algorithm to grow decision trees is given below:

```
Compute Entropy for dataset.
Select quantitative attributes.
For each attribute:
    Calculate entropy of all response variable.
    Calculate Mean entropy for the attribute.
    Calculate gain for the attribute.
Select the attribute with highest gain.
Repeat randomly after several iterations.
```

The entropy of a given attribute is calculated as follow:

$$S = -\sum_{i=1}^{N} p_i \, log_2(p_i) \tag{4.6}$$

The Information Gain (IG) for an attribute is calculated as:

$$IG(Q) = S_0 - \sum_{i=1}^{q} \frac{N_i}{N} S_i \tag{4.7}$$

However, in this study we have considered a regression problem. Hence our objective is to split the tree in a way that the Residual Sum of Squares (RSS) is minimal. So, we calculate the absolute mean to bifurcate the samples into two splits, each linking two child nodes. The child nodes iteratively calculate the mean for rest of the samples and the tree is grown until a desired model is achieved. Performance measures such as dept of the tree, number of splits and pruning ratio can be controlled by hyper-parameter tuning. This topic is excluded in this study as it requires advanced hands-on experience with Matlab toolbox ™and will be discussed in future contributions to the literature.

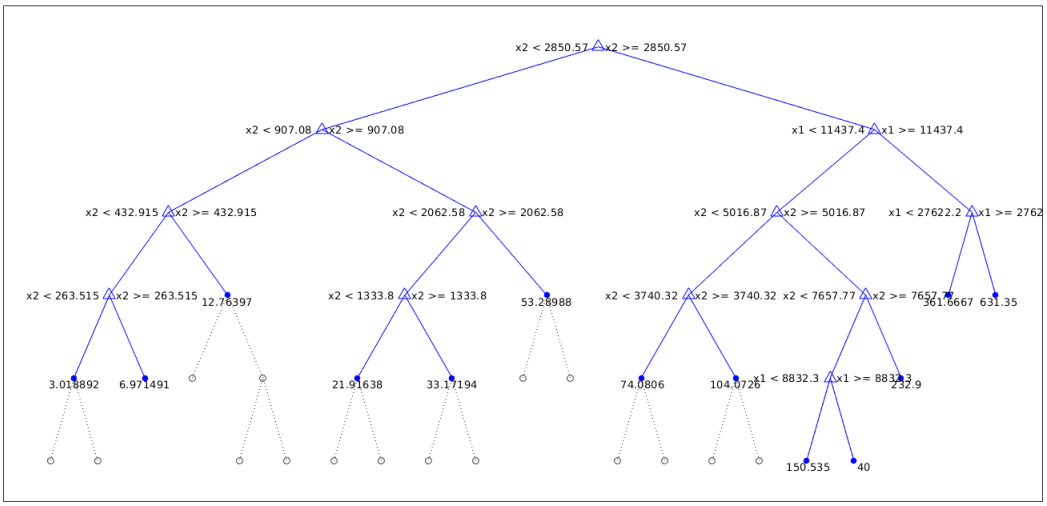
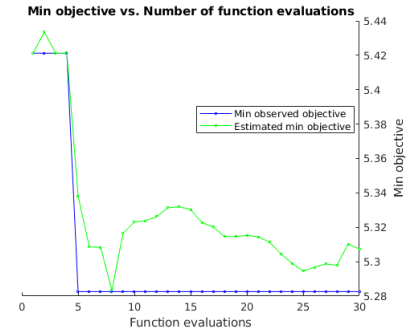
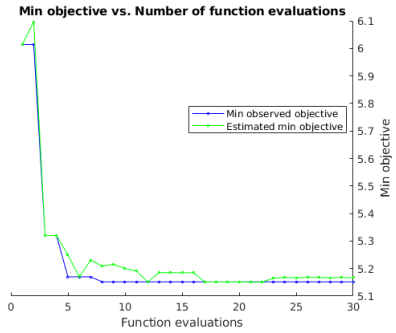


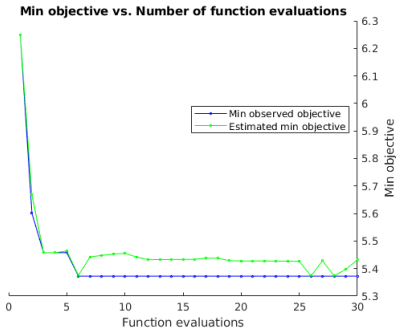
Figure 4.3: Decision Tree Model



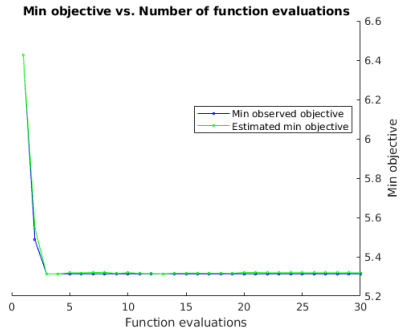
(a) Objective Function, Iteration 1



(b) Objective Function, Iteration 2



(c) Objective Function, Iteration 3



(d) Optimized Objective Function: Best Model

Figure 4.4: Plot for Minimum objective Function

4.2.3 Random Forest Regression

Decision trees are prone to variability since the selective features are arbitrarily drawn. Conducting a series of test on the same sample may result in different since individual

decision tree may tend to overfit. In order to overcome this anomaly, decision trees are combined to produce a collective result by combination or pooling, which reduced over-fitting and improves generalization.

Random Forests yields many additive decision tree and harvests a weighted predicted from all the trees. For a regression problem, the individual predictors produce their corresponding predictions and the mean prediction is considered as the final value.

Random Forests are built as follows:

```
Randomly select k attributes from dataset.
For k attributes, select best attribute (entropy selection and IG).
Split nodes into child nodes.
Repeat until maximum number of nodes 'l' are created.
Build forest for n number of times.
Predict for training samples.
Compute weighted average of individual predictions for final
    prediction.
```

The pseudo code for implementing Random Forest Regressors is also given below:

```

(trainingData, testData) = dataset.randomSplit([0.7, 0.3])

# Train a RandomForest model.
RFmodel = RandomForestRegressor(labelCol="Order_pure", featuresCol="
    InpVec", numTrees=3, maxBins=5000)
model = RFmodel.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)
#Evaluate Performance
evaluator = RegressionEvaluator(
    labelCol="Order_pure", predictionCol="prediction", metricName="
    rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE)= %g" % rmse)
```

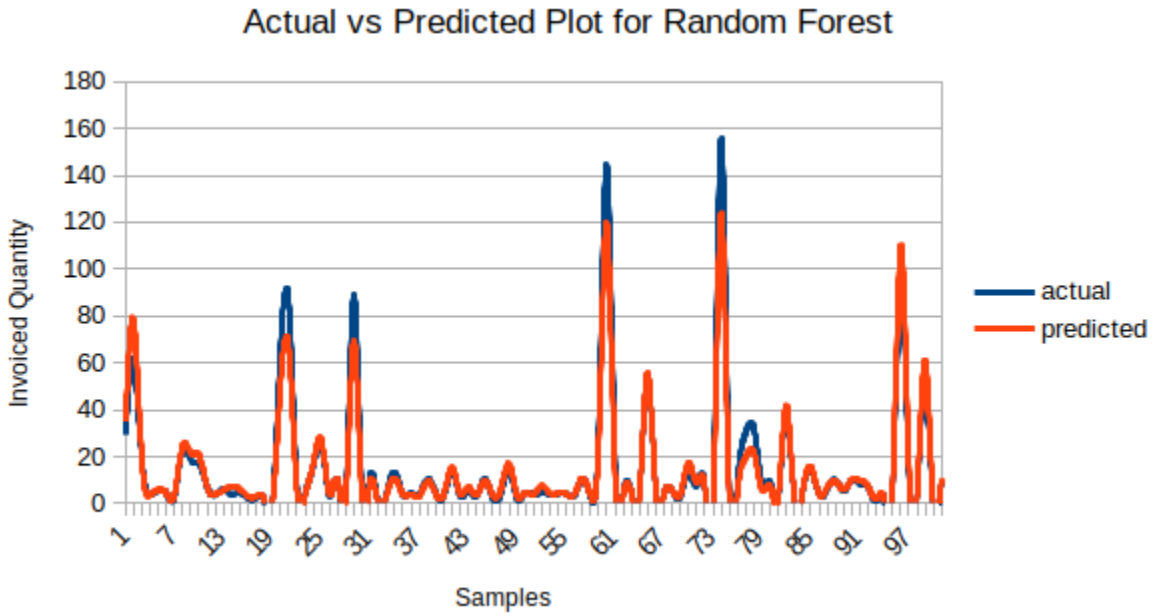


Figure 4.5: Actual vs Predicted plot for Random Forest predictors

4.2.4 XGB Tree Model

XGB Trees are ensemble learning approach with two or more hybrid learning algorithms. As discussed in the previous section, Decision Tree models exhibit high variance in generalization. Ensemble based learning approaches over this variance in a non-dominant fashion.

Boosting: Trees are generated sequentially such that each successive tree is emphasized to minimize the error generated by the preceding tree. The overall error is reduced since every generation of tree reduces error for its predecessors. In contrast to bagging techniques, in which trees are grown to their maximum extent, boosting uses trees with fewer splits. Several learning parameters such as number of trees or iterations, the learning rate or gradient boosting rate, dept of each tree can be optimally selected to reduce the computing overhead.

Boosting Algorithm:

```
1 Train initial model F_0 to predict target Y.
2 Compute residual error, delta_y = Y - F_0.
3 Create new model H_1 and fit to the residuals.
4 Combine (F_0 + H_1) to yield F_1.
```

```
Repeat for F_1
```

We create a model with a function F_0(x).

$$F\_0(x) = argmin \sum_{1}^{n} L(y_i, \gamma) \tag{4.8}$$

$$argmin \sum_{1}^{n} L(y_i, \gamma) = argmin \sum_{1}^{n} (y_i - \gamma)^2 \tag{4.9}$$

Taking the first differential w.r.t γ ,

$$F\_0(x) = \frac{1}{n} \sum_{1}^{n} (y_i) \tag{4.10}$$

The additive model H_1(x) computes the mean of the residuals at each leaf node of the tree. The boosted function F_1(x) is obtained by summing F_0(x) with H_1(x).

Numerical Example

Consider a regression problem with input feature Sales and target variable Quantity.

Sales	Quantity	F_0	y - F_0	H_1	F_1	y - F_1
5	82	134	-52	-38.25	95.75	-13.75
7	80	134	-54	-38.25	95.75	-15.75
12	103	134	-31	-38.25	95.75	7.25
23	118	134	-16	-38.25	95.75	22.25
25	172	134	38	25.50	159.50	12.50
28	127	134	-7	25.50	159.50	-32.50
29	204	134	70	25.50	159.50	44.50
34	189	134	55	25.50	159.50	29.50
35	99	134	-35	25.50	159.50	-60.50
40	166	134	32	25.50	159.50	6.50

Table 4.4: Tabulated performance of Additive Boosting Models

Pseudo code

```
### XGBRegressor Model
params = {'max_depth':2,
          'silent':0,
          'colsample_bytree':0.3,
          'max_dept':5
          'alpha':10
          'learning_rate':0.38,
          'objective':'reg:linear',
          'n_estimators':10}

features_x=np.array(output.select("vecFea").collect())
labels_y=np.array(output.select("QUANTITY").collect())
features_x=np.squeeze(features_x,axis=1)

X_train,X_test,Y_train,Y_test=model_selection.train_test_split(
    features_x,labels_y,test_size=0.51,random_state=123)

xgbmodel = xgb.XGBRegressor(params)
xgbmodel.fit(X_train,Y_train)
preds = xgbmodel.predict(X_test)
```

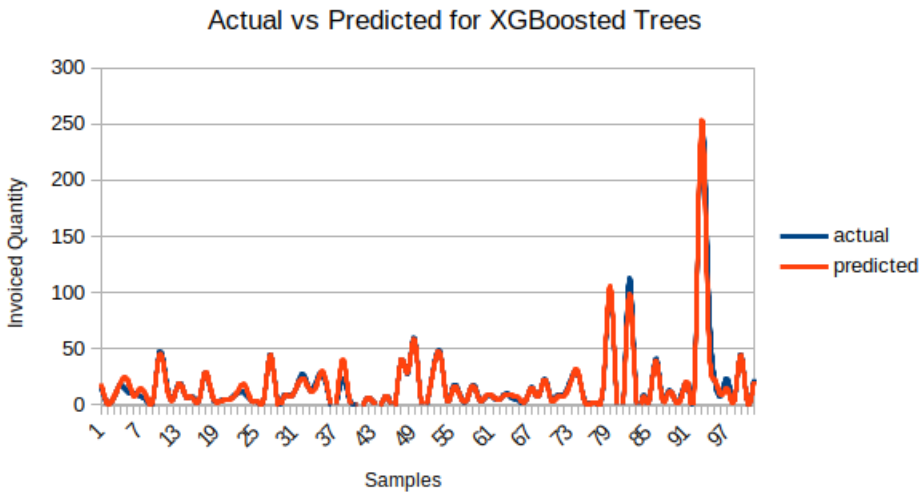


Figure 4.6: Actual vs Predicted Plot for XGBoosted Tree Model

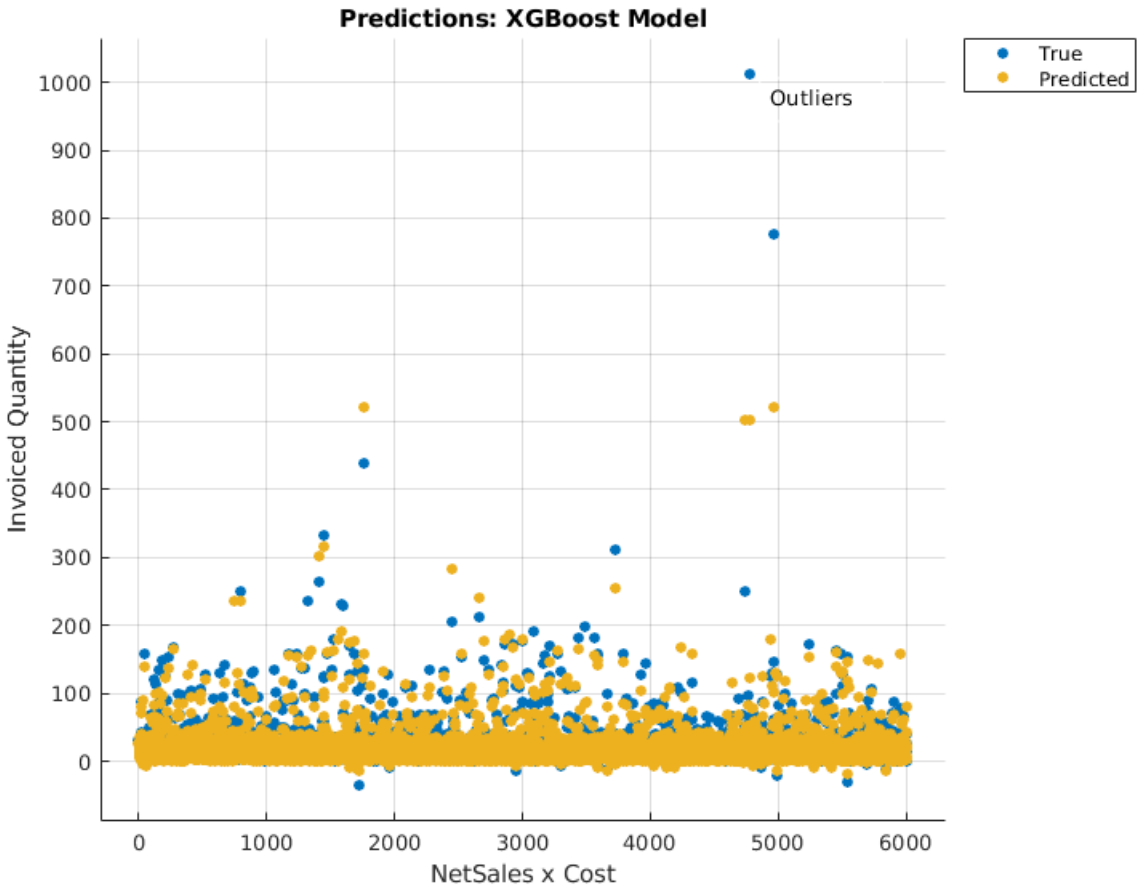


Figure 4.7: Response Plot for XGBoosted Trees

4.3 Performance Analysis

In this section, we will discuss the performance of each model based on empirical observations.

We attempted hyper-parameter tuning for the discussed models and overlooked optimization parameters to better understand the run-time performance of these model. The source code was compiled and executed with varying model parameters and the respective loss function was evaluated as a MILP optimization problem, the loss function being the sole objective function.

Table 4.5: Tabulated performances of ML Models

Model	Test loss	Tuning loss	RMSE	Max Dept	Trees Pruned	Training Time (in s)	Memory Utilized
Decision Tree (maxBins=100)	0.59	0.60	0.76	7	22	67	66B
RandomForest {n_estimators=50, MaxBins=100}	2.36	1.82	1.53	23	3	82	246B
RandomForest {n_estimators=100, MaxBins=100}	1.20	2.66	1.09	18	3	87	107B
XGB Tress {'colsample_bytree': 0.6, 'alpha': 10, 'learning_rate': 0.25, 'max_depth': 100, 'n_estimators':100}	7.90	7.28	2.81	38	7	1020	2081B
XGB Tress {'colsample_bytree': 0.3, 'alpha': 8, 'learning_rate': 0.38, 'max_depth': 50, 'n_estimators':50}	6.69	6.28	2.58	29	2	340	$\sim$1040B

As observed in 4.5, the respective performance metrics for each model is noted and tabulated. We achieve a low test error in Random Forest and Decision Tree model as compared to XGB models. This comparison is drawn based on accounting two hypothetical assumptions:

1. The relative error in the sample dataset is uniformly distributed.
2. The percentage of outliers are negligible in training phase.

Hyper-parameter tuning for XGB Tree models is performed extensively after carefully examining the dataset and its statistical parameters. We employ data visualization techniques to reduce internal dependency on stochastic constraints and estimate a number of predictors/estimators based on the average rate of change of the response variable. We achieve a decent score, but however, since the number of boosting rounds is only considered as 5, the optimization process is terminated after obtaining a satisfactory measurement. In context to learning parameters, we have noticed that a incremental change in the learning rate brings a linear change in the performance, and the topological difference in the model which played a central role in optimizing the objective function.

CHAPTER 5

NEURAL APPROACHES

5.1 Multi-layered Perceptron Networks

We carefully examine the neural network architecture ranging from number of hidden layers used and number of hidden neurons inside every layer. It is imperative to first select a distinctive problem and a neural network model in order to determine its model and run-time performance. For example, In our experiment we have considered the dataset "UAE_distributor.xlsx" in order to evaluate a Multi-Layered Perceptron (MLP) model and perform a function approximation to predict invoiced sales quantity.

5.1.1 Training the Model

The dataset contain 6011 samples containing the above mentioned attributes. We split the population samples as 80% training samples and 20% testing samples. We use the generalized back-propagation algorithm, as described in Appendix B. We then tabulate the observations after performing multiple experiments altering the chronology of training and randomizing the samples.

Table 5.1: Recorded Performance for MLP Network with 10 Hidden Layers

Model parameters		Comparative Measurement			
		Training measurement		Testing measurement	
Sample (%)	Layers	MSE	R	MSE	R
70	10	54.03	0.99082	61.69	0.99079
60	10	75.81	0.95061	77.40	0.91088
50	10	118.06	0.92061	127.88	0.9002

Table 5.2: Training performance of MLP Network

Network Architecture	Training Parameters					General Remarks
	Performance	Gradient	Mu	Epochs	Time	
MLP Network (1 Input Layer + 10 Hidden Layer + 1 Output Layer)	54.0	96.1	1.00	25	0.03	Accepted
	45.8	1.13e+0.3	1.00	10	0.01	Best Fit
	62.6	1.60e+0.3	10	35	0.05	Acceptable
	63.6	50.9	10	6	0.01	Acceptable

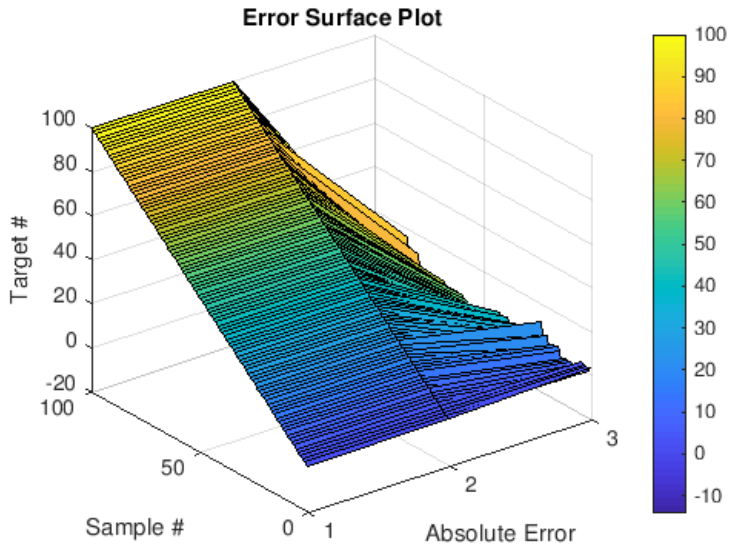


Figure 5.1: Error Surface Plot for Multi-Layered Perceptron Model

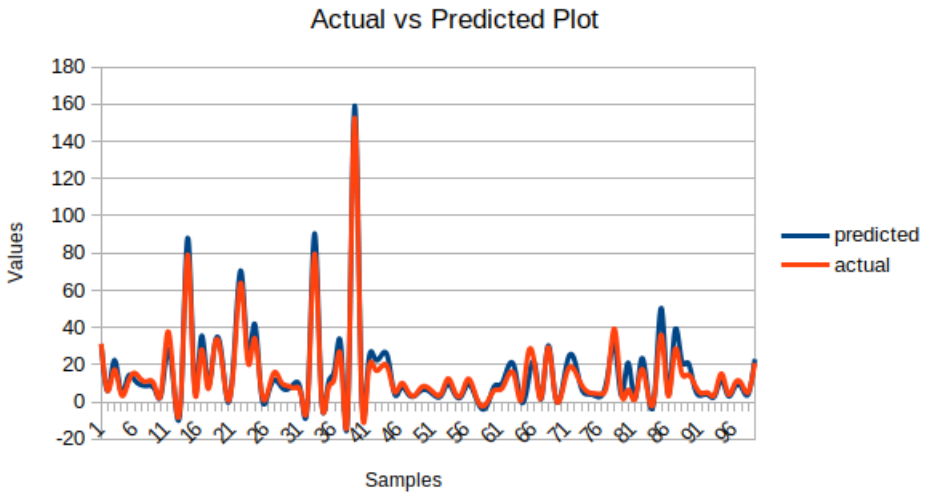


Figure 5.2: Actual vs Predicted Plot for MLP network with 10 Hidden Layers

5.2 Long-Short Term Memory (LSTM) Networks

LSTM is a type of artificial neural network designed especially for time series prediction problems. These have an input gate, output gate, forget gate and memory cell which are connected by loops adding feedback over time. The memory cell stores states which allow LSTMs to generalize patterns across large data points rather than succumbing to immediate patterns. As more layers of LSTM are added to the model, it is found to be successful across a diverse range of problems such as describing an image, grammar learning, music composition, language translation, etc. Unidirectional LSTM store information that has appeared in past whereas bidirectional LSTM store information from the past as well as the future in time series problems. It maintains two different hidden states for the same. Bidirectional LSTM is thus found more effective than the unidirectional variant as it understands the context better.

5.2.1 Architecture

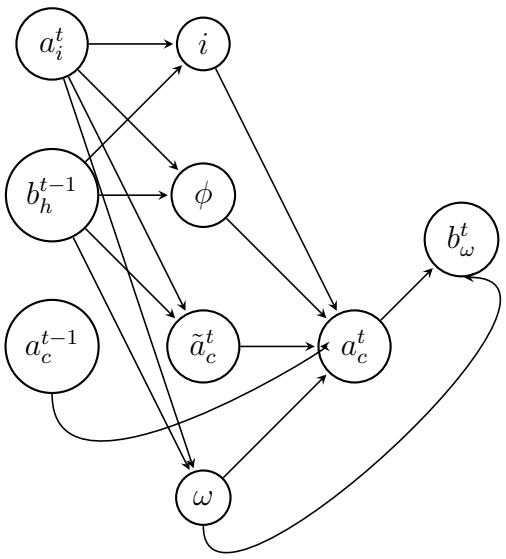


Figure 5.3: LSTM Architecture with one cell

Notations

- w_{ij} denotes connection weight from node i to node j

- a_i^t is network input to node j at a particular time t
- b_i^t value of node after applying activation function
- ι, ϕ and ω represents input gate, forget gate and output gate respectively
- C represents set of memory cells
- s_c^t denotes the state of a cell c at particular time t
- f represents activation function of gates, g denotes cell input activation functions whereas h is cell output activation functions
- I, K and H denotes number of inputs, number of outputs and number of cells in hidden layer respectively

Forward pass:

Input gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{5.1}$$

$$b_\iota^t = f(a_\iota^t) \tag{5.2}$$

Forget gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{5.3}$$

$$b_\phi^t = f(b_\phi^t) \tag{5.4}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{5.5}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{5.6}$$

Output gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^{t-1} \tag{5.7}$$

$$b_\omega^t = f(a_\omega^t) \tag{5.8}$$

Cell outputs

$$b_\omega^t = f(a_\omega^t) \tag{5.9}$$

Backward pass:

$$\epsilon_c^t = \frac{\partial O}{\partial b_c^t} \tag{5.10}$$

$$\epsilon_s^t = \frac{\partial O}{\partial s_c^t} \tag{5.11}$$

Cell outputs

$$\delta_\iota^t = f'(a_\iota^t) \sum_{c=1}^{C} g(a_c^t) \epsilon_s^t \tag{5.12}$$

Output gates

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^(t-1) \epsilon_s^t \tag{5.13}$$

States

$$\delta_c^t = b_\iota^t g'(a_c^t) \epsilon_s^t \tag{5.14}$$

Cells

$$\epsilon_s^t = b_\omega^t h'(s_c^t) \epsilon_c^t + b_\phi^(t+1) \epsilon_c^(t+1) + w_(c\iota) \delta_\iota^(t+1) + w_(c\omega) \delta_\omega^(t+1) \tag{5.15}$$

Forget Gates

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t) \epsilon_c^t \tag{5.16}$$

Input Gates

$$\epsilon_s^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{h=1}^{H} w_{ch}\delta_h^(t+1) \tag{5.17}$$

where $f(x) = \frac{1}{1+e^{-x}}$, $g(x) = \frac{4}{1+e^{-x}}$, $h(x) = \frac{2}{1+e^{-x}} - 1$ and $g(x)\epsilon[-2, 2]$, $h(x)\epsilon[-1, 1]$

5.2.2 Model

We used bi-directional LSTM to predict the selling price of items. Our feature vector consisted of selling price by the supplier, distributor, manufacturer and retailer. We used raw data of fluctuating prices every two minutes across 42 months starting from 9 Feb 2014 till 28 August 2017. The total number of data points were above 8,00,000. The model was trained on 70% of the data while 20% and 10% was used for testing and validation. Following diagram which represents the model summary.

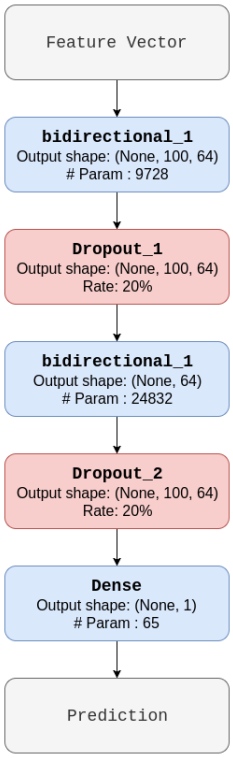


Figure 5.4: Model summary for Bi-LSTM Model

Following is error plot of the model after training.

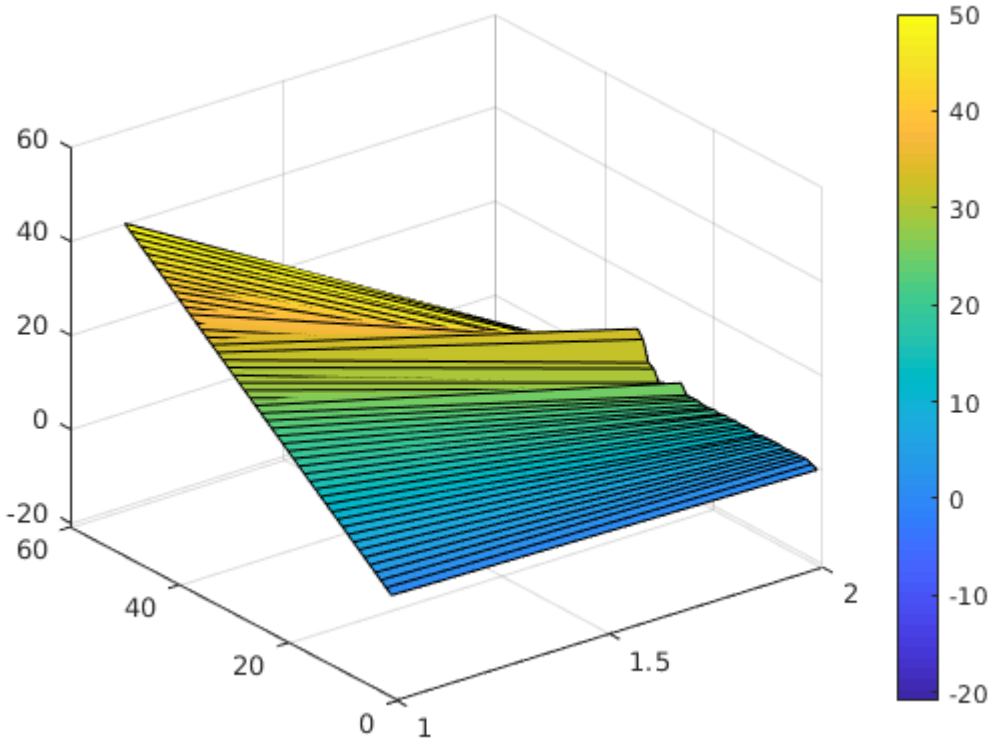
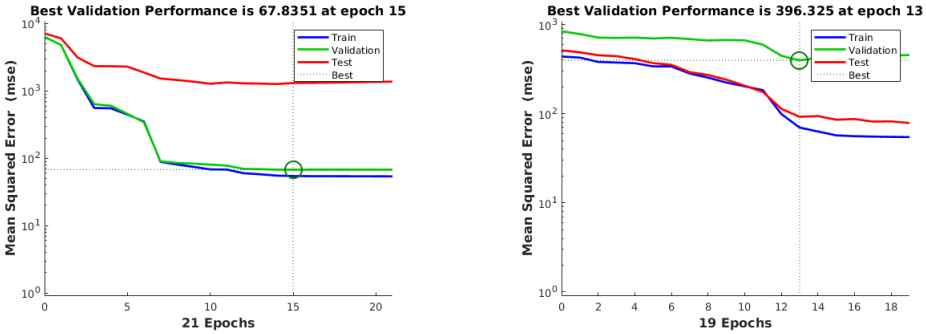


Figure 5.5: Error Surface plot for Bi-LSTM Model

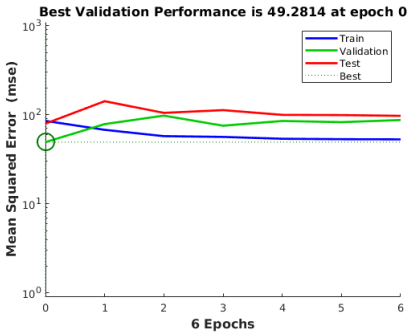
5.3 Performance Analyses

This section outlines the performance of each model extensively based on observations as well as empirical result. All the models were carefully optimized and tested for evaluating with benchmarks. We started with projecting the performance of each model in two-dimensional plots and attempted to deduce their inherent model performance by simple extrapolation. The principal objectives that is accounted for every model are the model parameters such as sample_size, number of hidden layers, number of neurons in each layer and the its effect on learning parameters. We attempted the analyses on obtained results from taken observations and extrapolated and extrapolated the model expectations further under the same learning parameters.



(a) Training cycle=1

(b) Training cycle=2



(c) Training cycle=3

Figure 5.6: Observed training performance of MLP networks after 3 training cycles

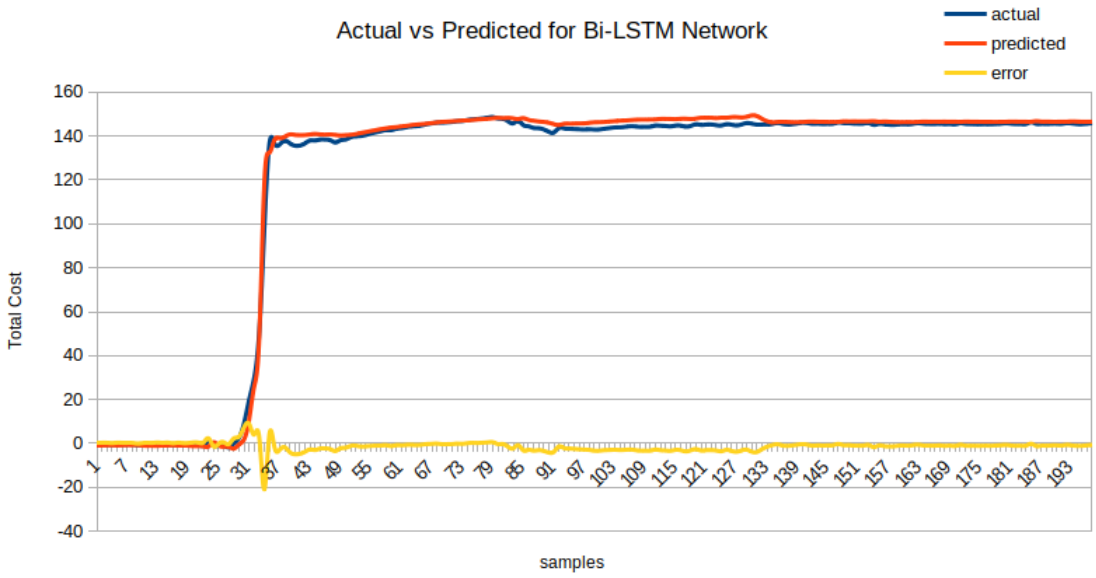


Figure 5.7: Actual vs Predicted plot for LSTM network

CHAPTER 6

TIME SERIES ANALYSES WITH MATLAB™

In this chapter, we will discuss problems related to time-series analysis specifically trend approximation and sequence-to-sequence prediction using Deep Learning Models. In the course of this study, we have generated a time-series sequence pertaining to risk-aware logistics relatable to a multi-echelon reverse supply chain network. The dataset is explicity described in Apprendix A.3 with mathematical notationsa and symbols. The objective in this case study is primarily towards building a nomenclature to reproduce the model evaluated and outlining the noteworthy bottlenecks encountered during the training and testing phase.

We have carefully investigated the time-series and after an initial visualization of the series, regular patterns were observed. The patterns suggested that the non-linear component instigating the trends in the time-series was coherent at every cycle with absolute minimal variations. The patterns continued till the end of the time-series and could be easily predictable as observed from a human vision. Enlarging the dataset, we encountered several regular patterns justifying that the time-series was masked under a linearly separable function without any substantial non-linear component acting on it. Hence, our conclusion from the initial study was that ANN models were also implementable for forecasting the series beyond its domain. However, for the sake of a comparative study, we have selected two distinctive time-series models and evaluated its performance to draw conclusions. We will discuss those models in the next section.

Broadly speaking, a time-series can be interpreted as series with a progression that have two components, one which is an independent component or linear component and the other which is a non-linear component as a function of time itself.

6.1 Non-Linear Auto-Regressive Model (NAR)

NAR Model is a predictive model that considers an input sequence y(t) of preceding 'd' time-steps to predicts the next sequence of d time-steps. This type of forecasting problems are termed as sequence-to-sequence prediction as the input sequence is coherently mapped with the target sequence.

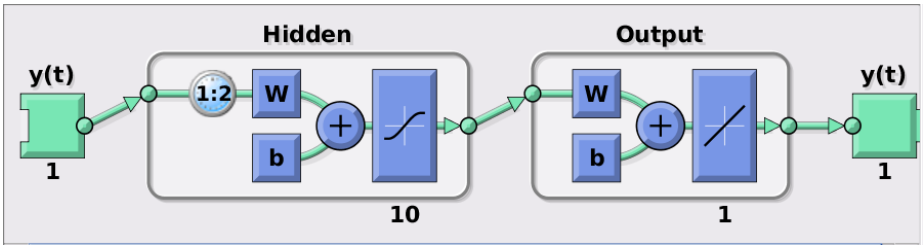


Figure 6.1: NAR Model Architecture

Mathematically, a general sequence-to-sequence problem can be represented as:

$$y(t) = f(y(t-1), y(t-2), y(t-3), ..., y(t-d)) \tag{6.1}$$

6.1.1 Pseudo Code

An example code is described to create a NAR model.

1. Importing a dataset as tabular data structures in Matlab ™.

```
Tinp=xlsread("<filname.xlsx", "<A2:A1000>");
Tinp'=csvread("filename.csv","<range>");
```

2. Time-series models do not operate on discrete data. They must be transformed into time-series sequence. Matlab™provides the preparets() function to convert discrete data into a continuous time-series.

```
NAR_model=narnet(1:3,5);
[Xs,Xi,Ai,Ts] = preparets(NAR_model,{},{},Tinp);
```

3. Every predictive model needs to be trained.

```
NAR_model = train(NAR_model,Xs,Ts,Xi,Ai);
```

4. Predict for series y(t) and evaluate performance.

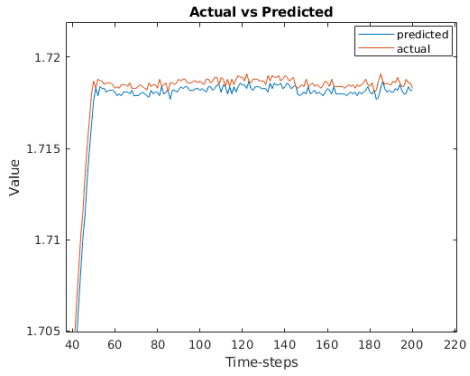
```
[Y,Xf,Af] = NAR_model(Xs,Xi,Ai);
perf = perform(NAR_model,Ts,Y)
```

Table 6.1: Training performance of NAR model

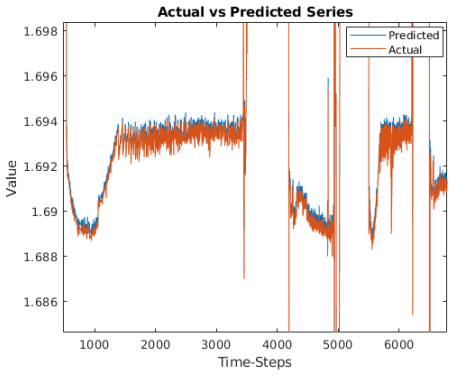
Network Architecture	Training Parameters				
	Performance	Gradient	Mu	Epochs	Time
NAR Model (delay 4 time-steps)	7.92e-05	1.67e-04	1.00e-06	12	0.04
	8.10e-05	5.18e-05	1.00e-06	18	0.04
	7.99e-05	1.48e-04	1.00e-07	6	0.01
	7.49e-05	2.88e-04	1.00e-07	14	0.04
	8.64e-05	1.44e-04	1.00e-07	20	0.05

[b] 7.92 e-05 denotes a very desirable model for sequence-to-sequence prediction.

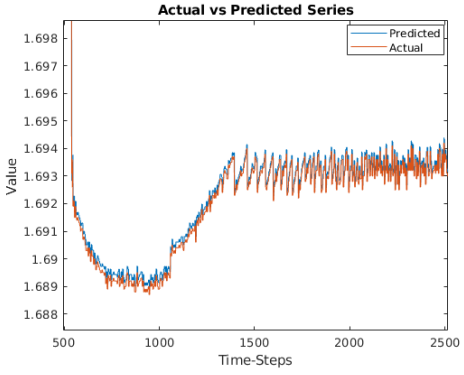
6.1.2 Results



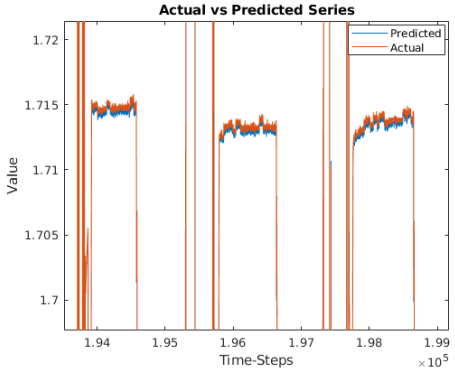
(a) NAR forecasted series 1



(b) NAR forecasted series 2



(c) NAR forecasted series 3



(d) NAR forecasted series 4

Figure 6.2: Actual vs Predicted series for NAR model

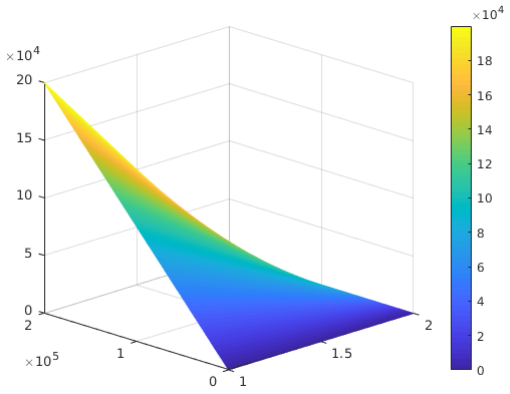


Figure 6.3: Error Surface Plot for NAR Model

Fig. 6.3 suggests that the error surface for NAR model is curvi-linear.The plot suggests a monotonous increase in the absolute error, however we need to calculate the rate of change of error to draw quantitative conclusion as compared to results obtained from other models. The rate of change of error can be derived as follows:

$$y = b\, e^{x}$$

$$\frac{dy}{dx} = b\, e^{x}$$

$$tan\theta = b\, e^{x}$$

$$\theta = tan^{-1}(b\, e^{x})$$

$$\frac{d\theta}{dx} = \frac{d}{dx}(tan^{-1}(b\, e^{x}))$$

$$\frac{d\theta}{dx} = \frac{1}{1 + b\, e^{2x}}$$

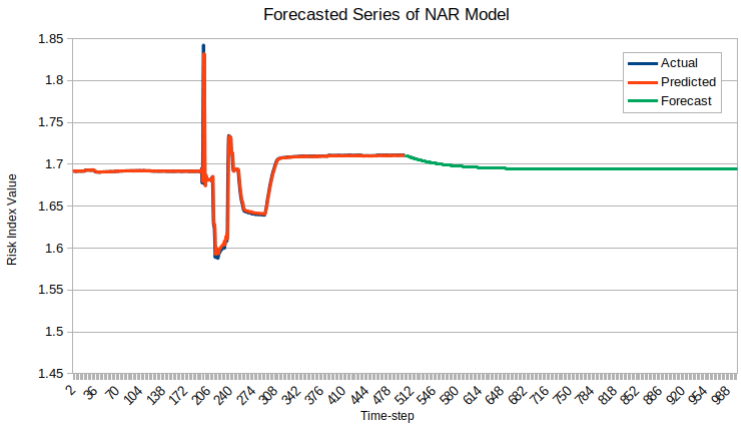


Figure 6.4: Forecasted Series for NAR Model

CHAPTER 7

CONCLUSION

The statistical hypothesis tests conducted across several datasets denotes arbitrary decisions on random sampling and data optimizations cannot be generally considered affirmative. As a contradictory remark, our hypothesis was rejected for the Walmart dataset. The tabulated results shows that even at 0.05% level of significance, the difference of mean of samples to that of population mean can vary and is in-determinant until calculated carefully. This study helped us to refrain from making hypothetical assumptions regarding the distribution of the dataset. As a conclusion, it is clear the the hypothesis validation would entirely depend on the distribution of the dataset and its degree of skweness.

Next, as we attempted several performance analyses on statistical machine learning and curve-fitting models, we studies the indigenous sparsity of the datasets. Even though some of the model reproduced identical series as compared to the actual series, the loss function could not be reduced detrimentally after a threshold. This signifies that even with multiple attempts of curve-fitting and hyper-parameter tuning, a stable model could not be achieved unless the underlying mathematical models are known. However, we also concluded that XGB model outperformed other models although Random Forest ensembles achieved a reasonably good fit. We also attempted hyper-parameter tuning for coarsed and simple decision tree model with random pruning and arbitrary number of splits. The objective function was optimized within 4 iterations, However, the Mean Squared Error (MSE) and gradient obtained after training was comparatively lower than the other models. We also compared Bi-directional LSTM networks with MLP networks.Bi-directional LSTM networks achieved outperforming results for sequence-to-sequence prediction, however the MLP networks failed to be robust against non-linear variations in the Time-series dataset. Finally, we discussed time-series analyses with NAR model and examined the results extensively. NAR model achieved outperforming MSE within the range of 1e-05 to 1e-06.

CHAPTER 8

CODE ANALYSIS

In this chapter, the program code related to our work using MATLAB™and Python is presented. Additionally, we have also provided a dynamic code analysis generated using the MATLAB™code analyzer.

8.1 Source Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 28 13:18:54 2019
Random Forest Regression with Pyspark
@author: heerokbanerjee
"""

import pandas as pd
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import Imputer
from pyspark.ml.feature import StringIndexer
from pyspark.sql.session import SparkSession
from pyspark.context import SparkContext

#
sc = SparkContext('local')
spark = SparkSession(sc)


#Importing Dataset
```

```
dataset = spark.read.format("csv").option("header","true").load("/home/heerokbanerjee/Documents/hpd.csv")
dataset = dataset.withColumn("Order_Demand",dataset["Order_Demand"].cast('double'))
dataset=dataset.select("Product_Code", "Warehouse", "Product_Category" , "Date","Order_Demand")

# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in ind
CodeIndexer= StringIndexer(inputCol="Product_Code", outputCol="CodeIndex",handleInvalid="skip")
WarehouseIndexer = StringIndexer(inputCol="Warehouse", outputCol="WarehouseIndex",handleInvalid="skip")
CategoryIndexer = StringIndexer(inputCol="Product_Category", outputCol="CategoryIndex",handleInvalid="skip")
DateIndexer= StringIndexer(inputCol="Date", outputCol="DateIndex", handleInvalid="skip")

assembler = VectorAssembler(
    inputCols=["CodeIndex", "WarehouseIndex", "CategoryIndex" , "DateIndex"],
    outputCol="Ghoda", handleInvalid="skip")

DemandImputer=Imputer(inputCols=["Order_Demand"], outputCols=["Order_pure"])

(trainingData, testData) = dataset.randomSplit([0.7, 0.3])
#trainingData.show()
# Train a RandomForest model.
rf = RandomForestRegressor(labelCol="Order_pure", featuresCol="Ghoda",
                           numTrees=3, maxBins=5000)


# Chain indexers and forest in a Pipeline
pipeline = Pipeline(stages=[CodeIndexer,WarehouseIndexer, CategoryIndexer,
                            DateIndexer,assembler,DemandImputer,rf])

```

```
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)

predictions.select("prediction").distinct().show()

predictions.distinct().show()

evaluator = RegressionEvaluator(
    labelCol="Order_pure", predictionCol="prediction", metricName="
    rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE)= %g" % rmse)


###plotting graph
eg = predictions.select("prediction","Order_pure","Ghoda").limit
    (1000)
panda_eg= eg.toPandas()

panda_eg.plot(kind="bar",stacked="true")
```

Listing 8.1: Random Forest Regression

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 23 02:12:41 2019
XGBoosted Classification with Pyspark and xgboost lib
@author: heerokbanerjee
"""

import numpy as np

from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer
#from pyspark.ml.classification import DecisionTreeClassifier
```

```
from pyspark.sql.session import SparkSession
from pyspark.context import SparkContext

from sklearn import model_selection
from sklearn.metrics import accuracy_score

import xgboost as xgb

sc = SparkContext('local')
spark = SparkSession(sc)

fname_train = "wallmart.csv"


def spark_read(filename):
        file = spark.read.format("csv").option("header", "true").load
    (filename)
        return file

def convert_to_numeric(data):
    for x in ["WEIGHT (KG)","MEASUREMENT","QUANTITY"]:
        data = data.withColumn(x, data[x].cast('double'))
    return data

### Import Training dataset
data = spark_read(fname_train)
data=data.select("ARRIVAL DATE","WEIGHT (KG)","MEASUREMENT","QUANTITY
    ","CARRIER CITY")
(train_data, test_data)=data.randomSplit([0.8,0.2])
train_data=convert_to_numeric(train_data)

### Pipeline Component1
### String Indexer for Column "Timestamp"
###
dateIndexer = StringIndexer(
        inputCol="ARRIVAL DATE",
        outputCol="dateIndex",handleInvalid="skip")
#print(strIndexer.getOutputCol())

```

```
#indexer_out.show()

### Pipeline Component2
### String Indexer for Column "Label"
###
carrierIndexer = StringIndexer(
        inputCol="CARRIER CITY",
        outputCol="carrierIndex",handleInvalid="skip")
#print(strIndexer.getOutputCol())
#out2 = labelIndexer.fit(train_data).transform(train_data)

### Pipeline Component2
### VectorAssembler
###
vecAssembler = VectorAssembler(
        inputCols=["WEIGHT (KG)","MEASUREMENT","QUANTITY","dateIndex"
    ],
        outputCol="vecFea",handleInvalid="skip")
#assembler_out = vecAssembler.transform(indexer_out)
#assembler_out.select("vecFea").show(truncate=False)

### Pipeline Component3
### GBT Classifier
#dt_class=DecisionTreeClassifier(labelCol="IndexLabel", featuresCol="
    vecFea")

### Training- Pipeline Model
###
pipe=Pipeline(stages=[dateIndexer,carrierIndexer,vecAssembler])
pipe_model=pipe.fit(train_data)

output=pipe_model.transform(train_data)
out_vec=output.select("dateIndex","vecFea").show(10)

num_classes=output.select("carrierIndex").distinct().count()
print(num_classes)
### XGBoostClassifier Model
###
###
```

```
89 params = {'max_depth':2,
90           'silent':0,
91           'learning_rate':0.38,
92           'objective':'multi:softprob',
93           'num_class':284}
94
95 features_x=np.array(output.select("vecFea").collect())
96 labels_y=np.array(output.select("carrierIndex").collect())
97 print(max(labels_y))
98 features_x=np.squeeze(features_x,axis=1)
99 X_train,X_test,Y_train,Y_test=model_selection.train_test_split(
       features_x,labels_y,test_size=0.51,random_state=123)
100
101
102 xgb_train = xgb.DMatrix(X_train, label=Y_train)
103 xgb_test = xgb.DMatrix(X_test, label=Y_test)
104
105 #xgbmodel=XGBClassifier()
106 xgbmodel = xgb.train(params, xgb_train,10)
107 print(xgbmodel)
108
109 ### Testing Pipeline + XGBoostClassifier
110 ###
111
112
113
114 test_output=pipe_model.transform(test_data)
115
116 xgb_output=xgbmodel.predict(xgb_test)
117 print(xgb_output)
118
119 predictions = np.asarray([np.argmax(line) for line in xgb_output])
120 print(predictions)
121
122 ### Determining Accuracy Score
123 ###
124 accuracy = accuracy_score(Y_test, predictions)
```

```
125 print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Listing 8.2: XGBoosted Classification

```
1  import numpy as np
2  import pandas as pd
3  from keras.models import Sequential
4  from keras.layers import Dropout
5  from keras.layers import LSTM
6  from keras.layers import Dense
7  from keras.layers import Bidirectional
8  from keras.callbacks import ModelCheckpoint
9  from sklearn.metrics import mean_squared_error as mse
10
11 input_file = "train.csv"
12 df = pd.read_csv(input_file)
13
14 def remove_duplicates(df):
15   ColoumnArr = np.array(df.ix[:, 'Timestamp'])
16   i=0
17   ArrLen = len(ColoumnArr)
18   index_duplicate= []
19   # identify duplicates by index
20   while(i<ArrLen-1):
21     if ColoumnArr[i]==ColoumnArr[i+1]:
22       index_duplicate.append(i+1)
23     i+=1
24   # remove duplicates
25   df=df.drop(index_duplicate)
26   return df
27
28
29 def avg_over_time(df, indexCol=0):
30   avg={}
31   colLen= df.shape[0]
32   for x in range(colLen):
33     time = str(df[x][indexCol])[11:]
34     if time not in avg:
35       avg[time]= [[0.0, 0.0, 0.0, 0.0, 0.0], 0]
36     for colNo in range(1,6):
```

```
      avg[time][0][colNo-1]+= float(df[x][colNo])
    avg[time][1]+=1
  for key, val in avg.items():
    avg[key]= [x*1.0/val[1] for x in val[0]]
  return avg


def replace_noise(df, indexCol=0):
  avg= avg_over_time(df, indexCol)
  colLen= df.shape[0]
  for x in range(colLen):
    time = str(df[x][indexCol])[11:]
    for col in range(1,6):
      if df[x][col]==0:
        try:
          df[x][col]= avg[time][col-1]
        except KeyError:
          print(x)
          print(col)
  return df

df= df.drop(['Label'], axis=1)
df= df.replace(np.nan, 0)
df= remove_duplicates(df).values
df= replace_noise(df)
print(df[0])

#data preperation
seq_length= 100
DataX= []
DataY= []

for x in range(len(df)-seq_length):
  SeqX= df[x: x+seq_length, 1:6]
  SeqY= df[x+seq_length, 5]
  DataX.append(SeqX)
  DataY.append(SeqY)

DataX=np.array(DataX)
```

```
DataY=np.array(DataY)

# transforming to (samples, seq length, features)
DataX= np.reshape(DataX, (DataX.shape[0], DataX.shape[1], DataX.shape
    [2]))
DataY= np.reshape(DataY, (DataY.shape[0], 1))
TrainDataX= DataX[:int(0.7*len(DataX))]
TrainDataY= DataY[:int(0.7*len(DataY))]
TestDataX= DataX[int(0.7*len(DataX)):]
TestDataY= DataY[int(0.7*len(DataY)):]

#developing the model
model = Sequential()
model.add(Bidirectional(LSTM(32, return_sequences=True), input_shape
    =(TrainDataX.shape[1],TrainDataX.shape[2])))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(32)))
model.add(Dropout(0.2))
model.add(Dense(TrainDataY.shape[1]))
filename="best_weight.hdf5"
model.load_weights(filename)
model.compile(loss='mean_squared_error', optimizer='adam')
#checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1,
    save_best_only=True,
#mode= min )
#callbacks_list = [checkpoint]
# fit the model
#model.fit(TrainDataX, TrainDataY, nb_epoch=50, batch_size=64,
    callbacks= callbacks_list)
print("loaded weights")
y = model.predict(TestDataX)
print("predicted")
print(mse(TestDataY, y))
```

Listing 8.3: Bi-LSTM Forecasting

8.2 Dynamic Code Analysis

We executed the MATLAB programs in MATLAB™online cloud platform and the Machine Learning models with standalone python library. Additionally, a series of benchmark tests were conducted to assess whether they can be implemented for extrapolated samples from the original datasets. The time complexities for each successive function call is given to trace the call stack.

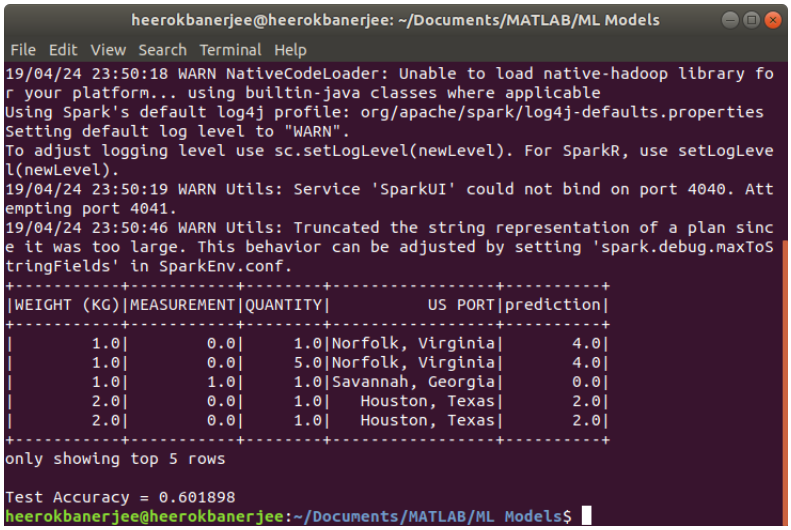


Figure 8.1: Execution of Code snippet for Decision Trees

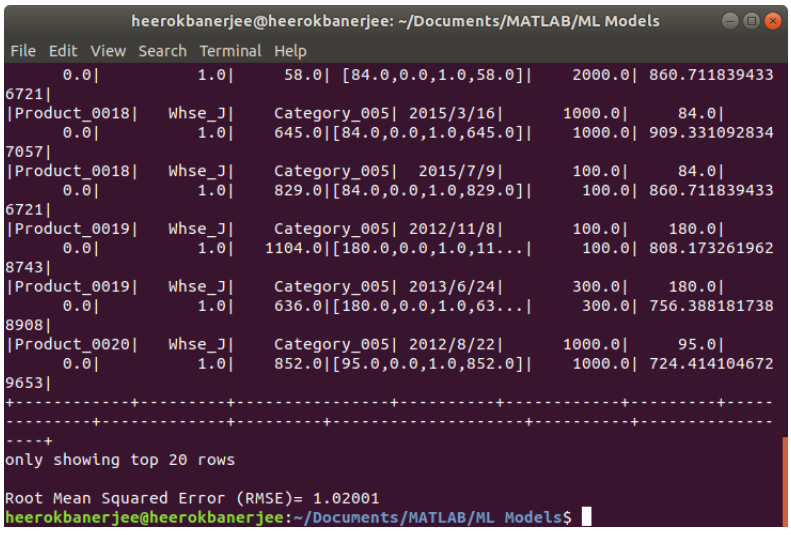


Figure 8.2: Execution of Code snippet for Random Forest

A static code analysis may be interpreted as a benchmark test to establish if the source code is syntactically accurate and is well-structured for the assembler to translate the program.

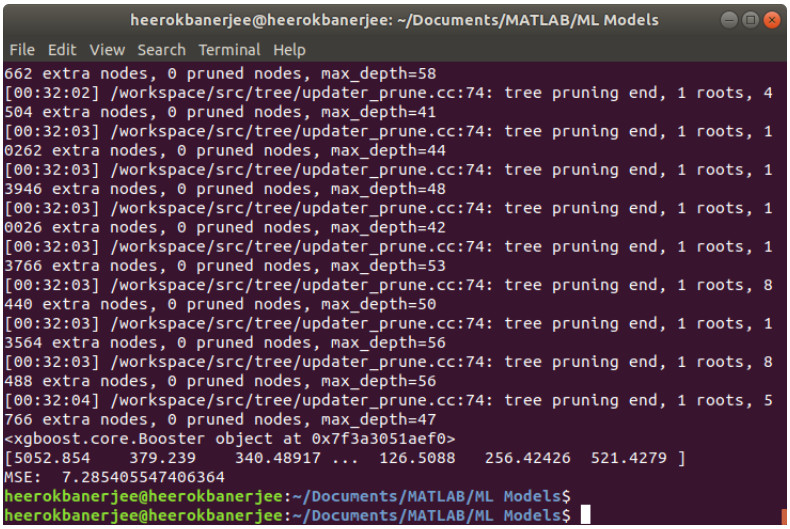


Figure 8.3: Execution of Code snippet for XGB Trees

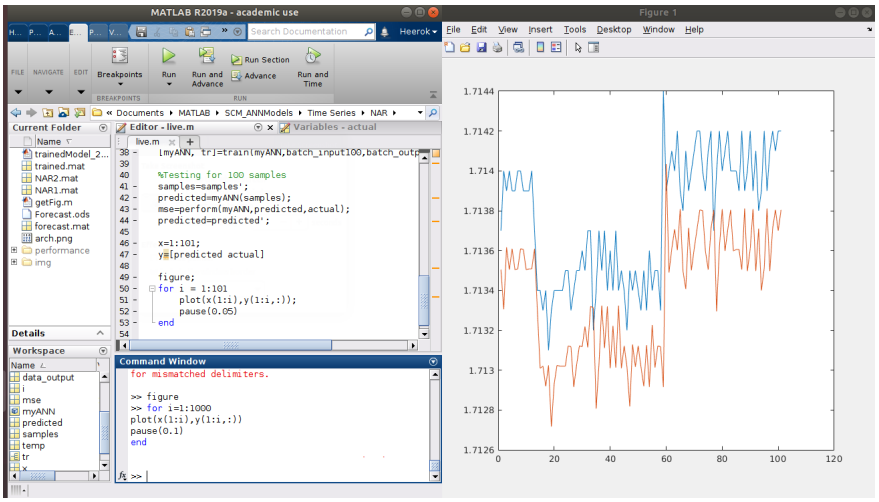
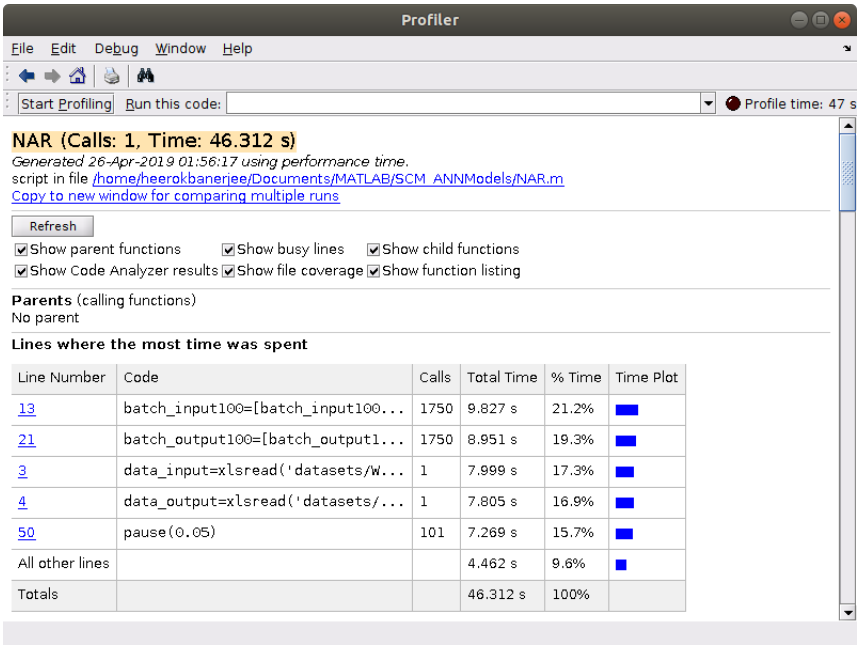


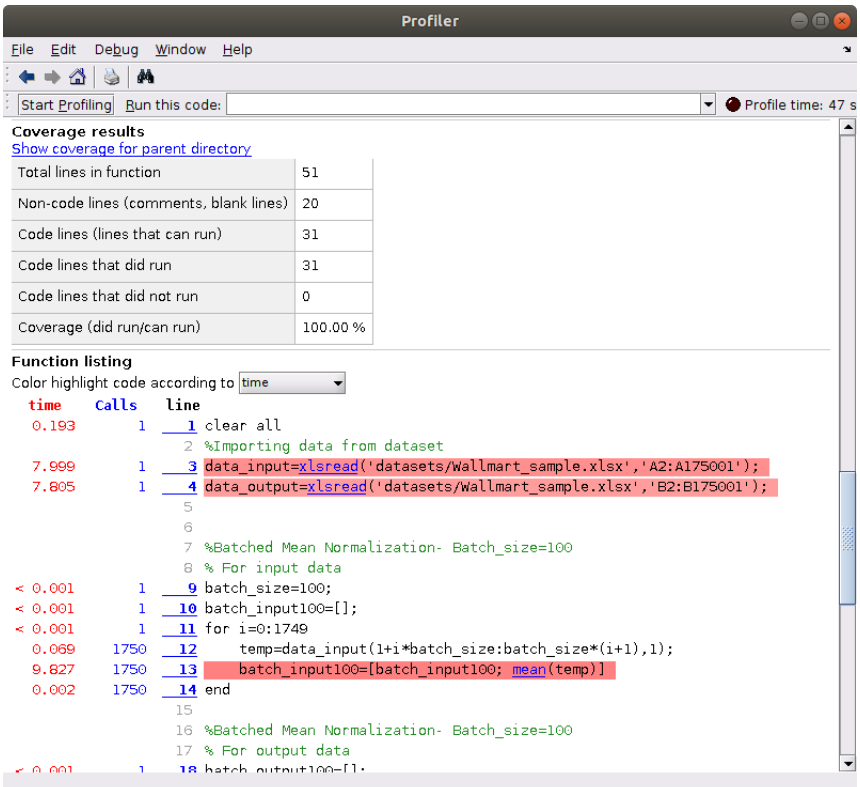
Figure 8.4: Execution of Code snippet for NAR live forecasting

Along with the execution of each model, we also attempted to understand the inherent time complexity and functional dependencies of our algorithmic steps. In order to evaluate a line-by-line dynamic code analysis and obtain a tabulated result, we have chosen MATLAB ™Code Analyzer to generate an automated report. The report depicts the total time elapsed for each functional call and allows to optimize a given source code in terms of modular programming standards.

We have considered individual time elapsed and cyclomatic complexity of different functions as a parameter while concluding our tests. The results highlight the section of the code with the dominant consumption of time due to recursive calls and higher-order operations.



(a) Breakdown of Total Elapsed Time for Function calls



(b) Time complexity and cyclomatic complexity

Figure 8.5: Dynamic code analysis using Matlab™Code Analyzer

8.3 Test Cases

In this section, we have tabulated the test cases and their respective dimensions as subjected to the prepared models.

Table 8.1: Test cases for ML models

Model	Dataset	Dimension	Model Parameters Dimenstions	
			Input Features	Output Feature
Decision Tree	Walmart.csv	[190962,33]	[50000,3]	[50000,1]
	UAE_Distributor.xlsx	[6011,7]	[2500,2]	[2500,1]
Random Forest	UAE_Distributor.xlsx	[6011,7]	[2500,1]	[2500,1]
	UAE_Distributor.xlsx	[6011,7]	[4500,3]	[4500,1]
	UAE_Distributor.xlsx	[6011,7]	[2500,3]	[2500,1]
XGBoosted Tree	Walmart.csv	[190962,33]	[150000,3]	[150000,1]
	Walmart.csv	[190962,33]	[100000,7]	[100000,1]

Table 8.2: Test cases for ANN Models

Model	Dataset	Dimension	Model Parameters Dimenstions	
			Input Features	Output Feature
MLP	Walmart.csv	[190962,33]	[150000,4]	[150000,1]
	UAE_Distributor.xlsx	[6011,7]	[5500,4]	[5500,1]
NAR	train.csv	[800001,7]	[80000,7]	[80000,1]
	train.csv	[800001,7]	[500001,7]	[500001,1]
Bi-LSTM	Walmart.csv	[190962,33]	[150000,4]	[150000,1]
	train.csv	[800001,7]	[150001,3]	[150001,1]

CHAPTER 9

PUBLICATION

We have simulated a multi-echelon closed-loop supply chain and generated a time-series sequence of 6 lakh timesteps for the open research community. The dataset is published in **Mendeley data library** under the affiliation of SRM Institute of Science & Technology, India. (https://data.mendeley.com/datasets/gystn6d3r4/2)

"Banerjee, Heerok; Saparia, Grishma; Ganapathy, Velappa; Garg, Priyanshi; Shenbagaraman, V. M. (2019), Time Series Dataset for Risk Assessment in Supply Chain Networks, Mendeley Data, v2 http://dx.doi.org/10.17632/gystn6d3r4.2"

Next, an article describing the mathematical modelling and a numerical example is published.

Grishma, Saparia: Time Series Dataset for Risk Assessment in Supply Chain Networks, ResearchGate, DOI: 10.17632/gystn6d3r4.2"

Finally, the results from this study is cross-validated with pre-existing models and we have finished drafting our research paper. We plan to submit the paper for a double-blinded peer review at Operations Research-PUBSonline library by April 2019.

APPENDIX A

DATASET DESCRIPTION

A.1 Walmart Dataset

The dataset "wallmart.csv" contains an extensive amount of data with specific geographical data like address, port description, destination details etc pertaining to the inherent logistics involved in a typical Walmart supply chain.

The dataset contains features such as: SHIPPER, SHIPPER ADDRESS, CONSIGNEE, CONSIGNEE ADDRESS, ZIPCODE, NOTIFY, NOTIFY ADDRESS, BILL OF LADING, ARRIVAL DATE, WEIGHT (LB), WEIGHT (KG), FOREIGN PORT, US PORT QUANTITY, Q.UNIT, MEASUREMENT, M.UNIT, SHIP REGISTERED IN, VESSEL NAME, CONTAINER NUMBER, CONTAINER COUNT, PRODUCT DETAILS, MARKS AND NUMBERS, COUNTRY OF ORIGIN, DISTRIBUTION PORT, HOUSE vs MASTER, MASTER B/L CARRIER CODE, CARRIER NAME, CARRIER ADDRESS, CARRIER CITY, CARRIER STATE, CARRIER ZIP, PLACE OF RECEIPT

A.2 UAE Distributor Dataset

The dataset "UAE_distributor.xlsx" stores an empirical record of a supply chain network. The data-sheet "Sales" contains 6011 samples describing sales transactions made between a single distributor and multiple customers across the UAE fish industry. Another data-sheet "Purchases" contains 94 samples representing purchases made by the distributor in procuring different fish products from multiple suppliers. The data-sheet "Items" describes the different items purchased and sold to different customers. The data-sheet "Customers" contains a list of customers. This dataset can be extensively utilized for regression problems.

The dataset contains 7 features such as :

1. Date (Timestamp)
2. Item Number (double)
3. Item Description (String)
4. Customer Name (String)
5. Net Sales (double)
6. Net Cost (double)
7. Average Sales Quantity (double)

A.3 Time Series Dataset

The Time-series sequence consists of 6 Lakh timesteps of seven attributes namely, Timestamp, RI_Supplier1, RI_Distributor1, RI_Manufacturer1, RI_Retailer1, Total_Cost, SCMstability_category. The time-series sequence is generated by simulating a multi-echelon supply chain network in MATLAB with three suppliers, one distributor, one manufacturer and one retailer. An arbitrary demand is introduced to the supply chain model and selective features such as total cost and risk index are calculated at each time-step.

The dataset is available online in Mendeley library.
To manually add the dataset in the bibiliography, use the following :
"Banerjee, Heerok; Saparia, Grishma; Ganapathy, Velappa; Garg, Priyanshi; Shenbagaraman, V. M. (2019), Time Series Dataset for Risk Assessment in Supply Chain Networks, Mendeley Data, v2 http://dx.doi.org/10.17632/gystn6d3r4.2"

APPENDIX B

BACKPROPAGATION ALGORITHM

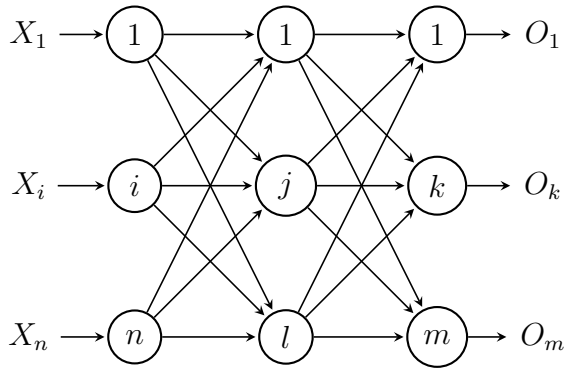


Figure B.1: Simple neural network architecture with one input layer of 'n' neurons, one hidden layer of 'l' neurons and one output layer of 'm' neurons

The error for he k^{th} neuron in the output layer is-

$$\delta_{O_k} = (T_{p_k} - O_k) \tag{B.1}$$

The error minimized by the Generalized Delta Rule (GDR):

$$E_p(m) = \frac{1}{2}\sum_{k=1}^{m} \delta_{O_k}^2 \tag{B.2}$$

Using the estimate of gradient descent along the error surface to determine the weight update, W_{ij} we get

$$\Delta W_{ij}(m) = -\eta \frac{\partial E_p(m)}{\partial W_{ij}} \tag{B.3}$$

The weight update equation is given by-

$$W(m+1)_{ij} = W_{ij}(m) + \Delta W_{ij} \tag{B.4}$$

The net input of Hidden Layer is given by-

$$net_{pj}^{h} = \sum_{i=1}^{n} (W_{ij} \,.\, X_{pi}) + b_j^h \tag{B.5}$$

Output of neuron 'j' in hidden layer is-

$$y_{pj} = f_j^h(net_{pj}^h) \tag{B.6}$$

The net output of Output Layer is given by-

$$net_{pk}^{o} = \sum_{j=1}^{l} (W_{jk}^{o} \,.\, y_{pj})^h + b_k^o \tag{B.7}$$

Output of the k^{th} neuron-

$$O_{pk} = f_k^o(net_{pk}^o) \tag{B.8}$$

Using equ.(B.1) to equ.(B.8), we derive the equation for updating output layer weights (between hidden layer & output layer).

$$\Delta W_{jk}(m) = -\eta \frac{\partial E_p(m)}{\partial W_{jk}}$$

$$\partial \frac{E_p(m)}{W_{jk}} = -\eta \frac{\delta E_p(m)}{\delta W_{jk}}$$

REFERENCES

[1] Ivanov, Dmitry, Alexander Tsipoulanidis, and Jörn Schönberger. "Basics of Supply Chain and Operations Manageme04nt." Global Supply Chain and Operations Management. Springer, Cham, 2017. 1-14.

[2] Van der Vorst, J. G. A. J. "Supply Chain Management: theory and practices." Bridging Theory and Practice. Reed Business, 2004. 105-128.

[3] Guedes, Edson Júnior Gomes, et al. "Risk Management in the Supply Chain of the Brazilian automotive industry." Journal of Operations and Supply Chain Management 8.1 (2015): 72-87.

[4] Flynn, Barbara, Mark Pagell, and Brian Fugate. "Survey research design in supply chain management: the need for evolution in our expectations." Journal of Supply Chain Management 54.1 (2018): 1-15.

[5] Scheibe, Kevin P., and Jennifer Blackhurst. "Supply chain disruption propagation: a systemic risk and normal accident theory perspective." International Journal of Production Research 56.1-2 (2018): 43-59.

[6] Ghadge, Abhijeet, et al. "A systems approach for modelling supply chain risks." Supply chain management: an international journal 18.5 (2013): 523-538.

[7] Ojha, Ritesh, et al. "Bayesian network modelling for supply chain risk propagation." International Journal of Production Research 56.17 (2018): 5795-5819.

[8] Santibanez-Gonzalez, Ernesto Del R., and Henrique Pacca Luna. "An Evolutionary Scheme for Solving a Reverse Supply Chain Design Problem." Proceedings on the International Conference on Artificial Intelligence (ICAI). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[9] Neureuther, Brian D., and George Kenyon. "Mitigating supply chain vulnerability." Journal of marketing channels 16.3 (2009): 245-263.

[10] Käki, Anssi, Ahti Salo, and Srinivas Talluri. "Disruptions in supply networks: A probabilistic risk assessment approach." Journal of Business Logistics 36.3 (2015): 273-287.

[11] Scheibe, Kevin P., and Jennifer Blackhurst. "Supply chain disruption propagation: a systemic risk and normal accident theory perspective." International Journal of Production Research 56.1-2 (2018): 43-59.

[12] Mastrocinque, Ernesto, et al. "A multi-objective optimization for supply chain network using the bees algorithm." International Journal of Engineering Business Management 5.Godite 2013 (2013): 5-38.

[13] Cao, Cejun, et al. "A novel multi-objective programming model of relief distribution for sustainable disaster supply chain in large-scale natural disasters." Journal of Cleaner Production 174 (2018): 1422-1435.

[14] Hajikhani, Alborz, Mohammad Khalilzadeh, and Seyed Jafar Sadjadi. "A fuzzy multi-objective multi-product supplier selection and order allocation problem in supply chain under coverage and price considerations: An urban agricultural case study." Scientia Iranica 25.1 (2018): 431-449.

[15] Loni, Parvaneh, Alireza Arshadi Khamseh, and Seyyed Hamid Reza Pasandideh. "A new multi-objective/product green supply chain considering quality level reprocessing cost." International Journal of Services and Operations Management 30.1 (2018): 1-22.

[16] Cao, Cejun, et al. "A novel multi-objective programming model of relief distribution for sustainable disaster supply chain in large-scale natural disasters." Journal of Cleaner Production 174 (2018): 1422-1435.

[17] Cao, Cejun, et al. "A novel multi-objective programming model of relief distribution for sustainable disaster supply chain in large-scale natural disasters." Journal of Cleaner Production 174 (2018): 1422-1435.

[18] Singh, Sujeet Kumar, and Mark Goh. "Multi-objective mixed integer programming and an application in a pharmaceutical supply chain." International Journal of Production Research 57.4 (2019): 1214-1237.

[19] Metiaf, Ali, et al. "Multi-objective Optimization of Supply Chain Problem Based NSGA-II-Cuckoo Search Algorithm." IOP Conference Series: Materials Science and Engineering. Vol. 435. No. 1. IOP Publishing, 2018.

[20] Hendalianpour, Ayad, et al. "A linguistic multi-objective mixed integer programming model for multi-echelon supply chain network at bio-refinery." EuroMed Journal of Management 2.4 (2018): 329-355.

[21] Park, Kijung, Gül E. Okudan Kremer, and Junfeng Ma. "A regional information-based multi-attribute and multi-objective decision-making approach for sustainable supplier selection and order allocation." Journal of Cleaner Production 187 (2018): 590-604.

[22] Ebrahimi, M., R. Tavakkoli-Moghaddam, and F. Jolai. "Bi-objective Build-to-order Supply Chain Problem with Customer Utility." International Journal of Engineering 31.7 (2018): 1066-1073.

[23] Cao, Cejun, et al. "A novel multi-objective programming model of relief distribution for sustainable disaster supply chain in large-scale natural disasters." Journal of Cleaner Production 174 (2018): 1422-1435.

[24] Yildizbai, Abdullah, et al. "Multi-level optimization of an automotive closed-loop supply chain network with interactive fuzzy programming approaches." Technological and Economic Development of Economy 24.3 (2018): 1004-1028.

[25] Çalk, Ahmet, et al. "A Novel Interactive Fuzzy Programming Approach for Optimization of Allied Closed-Loop Supply Chains." International Journal of Computational Intelligence Systems 11.1 (2018): 672-691.

[26] Park, Kijung, Gül E. Okudan Kremer, and Junfeng Ma. "A regional information-based multi-attribute and multi-objective decision-making approach for sustainable supplier selection and order allocation." Journal of Cleaner Production 187 (2018): 590-604.