ALLOCATING COSTS IN A LOT SIZING GAME USING NOVEL MACHINE LEARNING METHODS

A Thesis

by

Furkan Kasapoğlu

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Data Science

Özyeğin University
May 2022

ALLOCATING COSTS IN A LOT SIZING GAME USING NOVEL MACHINE LEARNING METHODS

Approved by:

Assoc. Professor Okan Örsan Özener,
Advisor
Department of Industrial Engineering
Özyeğin University

Assoc. Professor Uğur Çelikyurt
Department of Finance
Koç University

Asst. Professor Başak Altan Özener
Department of Economics
Özyeğin University

Date Approved: 25 May 2022

To My Family

ABSTRACT

In supply chain management (SCM), effective resource utilization is the key to achieving certain strategic benefits such as minimizing costs, increasing service levels, reducing inventories, increasing responsiveness, and finally improving customer satisfaction. Collaborative approaches among supply chain entities have become increasingly popular to increase resource utilization. In this thesis, we analyze a collaborative production setting where several companies facing varying demands throughout a finite planning horizon attempt to reduce their procurement costs by ordering from a common supplier. As the capacity of the common supplier is better utilized in such a collaborative solution, it will yield benefits that will be shared by the collaborators. Our objective is to design a cost allocation framework to ensure the sustainability of the collaborative purchasing organization. We propose various methods, including novel and computationally efficient machine learning based methods, using two different architectures, gradient boosting mechanism, and artificial neural networks which ensure the scalability of the proposed framework. We perform an extensive computational study and observe that our proposed method significantly outperforms the generic methods in the literature in terms of solution quality and computation time.

Keywords: Economic Lot Sizing Problem; Capacity; Cost Allocation; Backorder; Machine Learning; Cooperative Game Theory

ÖZETÇE

Tedarik zinciri yönetiminde (TZY), etkin kaynak kullanımı, maliyetlerin en aza indirilmesi, hizmet düzeylerinin artırılması, stokların azaltılması, yanıt verme hızının artırılması ve son olarak müşteri memnuniyetinin artırılması gibi belirli stratejik faydaların elde edilmesinin anahtarıdır. Tedarik zinciri kuruluşları arasındaki işbirlikçi yaklaşımlar, kaynak kullanımını artırmak için giderek daha popüler hale geldi. Yapılan bu tez çalışması ile sınırlı bir planlama ufku boyunca değişen taleplerle karşılaşan birkaç şirketin ortak bir tedarikçiden sipariş vererek tedarik maliyetlerini düşürmeye çalıştığı ortak bir üretim ortamını analiz ediyoruz. Ortak tedarikçinin kapasitesi böyle bir işbirlikçi çözümde daha iyi kullanıldığından, işbirlikçiler tarafından paylaşılacak çeşitli faydalar sağlayacaktır. Amacımız, ortak satın alma organizasyonunun sürdürülebilirliğini sağlamak için bir maliyet dağıtım çerçevesi tasarlamaktır. Önerilen çerçevenin ölçeklenebilirliğini sağlamak için iki farklı algoritma, gradient boosting ve yapay sinir ağları, kullanan yeni ve hesaplama açısından verimli makine öğrenimi tabanlı yöntemler de dahil olmak üzere çeşitli yöntemler öneriyoruz. Bahsedilen yöntemlerle, kapsamlı bir hesaplama çalışması gerçekleştiriyoruz ve önerilen yöntemimizin, çözüm kalitesi ve hesaplama süreleri açısından literatürdeki jenerik yöntemlerden önemli ölçüde daha iyi performans gösterdiğini gözlemliyoruz.

Anahtar Kelimeler: Ekonomik Parti Büyüklüğü Problemi; Kapasite; Maliyet Dağılımı; Sipariş Bakiyesi; Makine Öğrenmesi; İşbirlikli Oyun Teorisi

ACKNOWLEDGEMENTS

First of all, I would like to say thank you to my advisor Okan Örsan Özener for his support and guidance during my data science education. His knowledge and experience in the subject guided me through my research. It was an honor to study under his guidance. In addition, I am grateful to my thesis committee for their contributions and suggestions on this thesis. I would like to say that I am grateful to my family, especially to my mother Nurgül Kasapoğlu and my father Şenol Kasapoğlu for their continuous and unconditional love and support. They are my inspirations throughout my life. Finally, thank you to our loved ones who rest in peace for all contributions to who I am.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In supply chain management (SCM), effective resource utilization is the key to achieving certain strategic benefits such as minimizing costs, increasing service levels, reducing inventories, increasing responsiveness, and finally improving customer satisfaction. There exist several initiatives implemented in SCM to make the most of available resources including material, labor, machinery, vehicles, energy, and capital. The main focus of all of these initiatives is to eliminate the operational inefficiencies caused by the characteristics of supply chain operations such as demand fluctuations, capacity restrictions, uncertainties in manufacturing/transportation/economical conditions, and information asymmetries among supply chain partners, etc. While some of these initiatives can be undertaken by a single supply chain entity, the others require a coordinated effort between supply chain partners in order to realize the benefits.

Collaborative procurement/group purchasing organizations (especially in healthcare supply chains), airline alliances, road transportation alliances, and ocean shipping alliances are some examples of coordination among supply chain partners. From both practical and theoretical perspectives, it has been proven that such collaborative initiatives offer significant potential benefits to the participants. As in all such initiatives, there is a synergistic gain such as more efficient utilization of the production/transportation capacity, better management of inventories, better planning against uncertainties, etc., which supposedly lead to the famous win-win type equilibrium in strategic interactions among the individuals. Unfortunately, this is only one side of the coin. The other side of the coin is in such collaborations there are several coordination issues, conflicts of interests, and trust issues among participants, which might limit the potential benefits or even

destroy them entirely. For instance, a group purchasing organization in essence aims to better utilize the production capacity of a common supplier, which leads to lower procurement costs for the participants. However, at the same time, these companies might battle for the same common capacity especially when the production capacity of the supplier is rather limited and the demand must be met in a timely manner. Therefore, a successful coordination mechanism to govern such collaborative operations is crucial to keep such collaborations intact.

One important aspect of designing such a coordination mechanism is to keep all participants financially satisfied as it boils down to increasing profits for most supply chain stakeholders and potential conflicts arise mostly due to monetary issues. This might seem straightforward at first as the collaboration will offer financial benefits to the entire collaboration. Since there exists such a surplus over the non-collaborative alternative, it should be relatively easy to identify common grounds to keep every participant content in the collaboration. However, it turns out that it is in fact not the case in many supply chain coordinations. The reason is mainly the dynamics/characteristics of the supply chain operations and potential coordination opportunities. For instance, while trying to utilize a common truck capacity in milk run type freight transportation, conflicts might arise among collaborators on the fraction of capacity used by each customer or on the order of deliveries to the customers. Similarly, in airline alliances one flight leg might be in high demand due to seasonality, flight network characteristics, etc. and several members of the alliances might request to book seats on that particular flight. However, the operating airline can only supply a limited number of seating due to the aircraft's capacity restriction. In such cases, a collaborative solution might be appealing to some participants whereas the others might reject such solutions. The key is to identify a solution that all participants agree to and yields positive benefits to overall collaboration.

In this thesis, we analyze a collaborative production setting where several companies which are facing varying demands throughout a finite planning horizon

attempt to reduce their procurement costs by ordering from a common supplier. As the capacity of the common supplier is better utilized in such a collaborative solution, it will yield benefits that will be shared by the collaborators. The underlying optimization problem that determines the minimum joint cost solution for the collaboration is referred to Joint Economic Lot Sizing Problem in the literature. The joint costs include the fixed ordering costs associated with each order, the variable cost of production, inventory holding costs, and finally backordering costs of unfulfilled demand. This multi-period production planning problem is a complex problem and proved to be NP-Hard under production capacity restrictions. The difficulty of the problem is further increased when cost values are also non-stationary besides the demand values, as the timing of the productions becomes much more critical.

Provided that the most cost-effective solution to the collaborative purchasing problem is determined, the next step is to share the responsibilities among the participants. Unfortunately, this is where the conflicts among the participants threaten the sustainability of the collaboration. The presence of the fixed cost and the non-stationary nature of the problem significantly contribute to these conflicts as the participants prefer to be least responsible for the inefficiencies of the purchasing process such as inventory holding/backorder costs and a portion of the fixed costs. Ideally, any company would prefer not to pay inventory holding/backorder costs however, this may not be possible due to the production capacity constraints and in fact, it may be even preferable due to the non-stationary production costs (if production costs in the previous and subsequent periods are so low that the companies might prefer keeping inventories or backordering the customer demand). Inclusion of the possibility of backordering, the demand backorder complicates the underlying optimization problem as the solution space becomes larger, however from the perspective of cost allocation having more options generally leads to better solutions.

Our objective is to design a cost allocation framework to ensure the sustainability of the collaborative purchasing organization. We model the problem as a cooperative game among the retailers with a common goal of minimizing the total purchasing cost. In line with the related literature, when allocating cost responsibilities we consider two major criteria: (i) total cost is shared among the participants (budget balanced allocation), and (ii) no participant has a better option for individual collaborative (stable allocation). In the literature, cooperative lot sizing games have been studied in varying settings. Most of these works assume an uncapacitated production process where a budget balanced and stable (which is referred to as core) allocation can be identified rather trivially. Under a production capacity restriction, such solutions may not exist, as participants might battle for the same production capacity based on the relative cost figures.

The main contribution of this work is the scalability of the allocation frameworks. In the literature, several generic allocation methods have been proposed for economic games such as the Shapley value and nucleolus that require extensive computations limiting their scalability. Therefore, we also propose novel and computationally efficient machine learning based methods, using two different architectures, gradient boosting mechanism, and artificial neural networks. These machine learning approaches enable us to predict the synergies of sub-coalitions without explicitly computing them and hence significantly improve the scalability of the cost allocation framework.

Machine learning based methods have been proven quite powerful in prediction especially when the right method is chosen among a variety of alternative methods. Nevertheless, it is also important to note that predicting the optimal objective function value of a combinatorial optimization problem presents a much greater challenge compared to the regular machine learning prediction tasks such as customer churn prediction, and house price estimation. The real challenge lies within the fact that in a combinatorial optimization setting the optimality conditions are based on a very intrinsic relationship based on the problem parameters,

constraints, and objectives. Therefore, it is relatively difficult to represent these in a learning-based framework even though a very complex and non-linear model is used in the learning process. Motivated by this, two completely different architectures and even include some variations in them to design a well-performing prediction routine are developed.

The remainder of this thesis is organized as follows: in Chapter 2, the related literature will be reviewed. In Chapter 3, the problem will be defined, the dataset will be introduced and the model results will be compared as a computational study. Concluding remarks are provided in Chapter 4.

CHAPTER II

LITERATURE REVIEW

In this chapter, we briefly review the related work in the literature. There are three main streams of literature that are relevant to our work: studies on lot sizing problem, cooperative games, and finally machine learning methods.

2.1 Lot Sizing Problem

There is a large body of literature on the lot sizing problem since its introduction [1]. Even though the base setting considering fixed, variable, and holding costs is proven to be polynomially solvable, the introduction of the capacity restriction, as is the case in our setting, makes the problem variant NP-Hard [2], [3]. Besides the capacity constraint, several additional properties/restrictions have been studied in the literature (backordering, inventory capacity, timing restrictions, perishability, multi-products, etc.). We refer the reader to Brahimi et al. [4], Buschkuhl et al. [5] and Brahimi et al. [6] for thorough reviews of lot sizing problem variants and the proposed solution methodologies.

2.2 Cooperative Games

The next stream of literature closely related to our work is the literature on cooperative games. Cooperative games refer to the strategic interactions among a group of individuals (players) that put a coordinated effort to achieve a common objective (i.e. maximization of the total profit, minimization of the joint costs). In that aspect, the entire set of players is referred to as the collaboration (or the grand coalition) and the subsets of the collaboration are referred to as coalitions.

In the literature, the studies on the cooperative game theory methods applied to the cooperative dynamic lot sizing problem are the closest studies to our work even though the number of such studies is rather limited. Van den Heuvel et al.

[7] is the first study to analyze the economic lot sizing game with fixed costs. The authors consider that a group of retailers that come together and order the same product from a common supplier and show that the game is balanced and that the core allocation set of this game is non-empty, by using the Bondareva-Shapley theorem. Xu and Yang [8] propose a cross-monotonic and approximate budget balanced cost-sharing method for a collaborative lot sizing game and numerically demonstrate the effectiveness of the proposed method. In this study, there is an option of backordering of the demand. Gopaladesikan et al. [9] examine the collaborative lot sizing game under general concave order cost functions and propose a combinatorial primal-dual algorithm in order to find a solution in the nucleolus in polynomial time. Toriello and Uhan [10] propose a method to compute a dynamic cost allocation in the strong sequential core of the game under concave ordering costs and finite planning horizon. The definition of a strong sequential core corresponds to the allocation that is budget balanced and stable over time. Chen and Zhang [11] show for the collaborative lot sizing game under general concave costs and backordering always has a non-empty core and such an allocation can be computed in polynomial time by solving a modified dual problem. They also prove that not all dual solutions correspond to a solution in the core contrary to the similar results in the literature. Finally, Drechsel and Kimms [12] examine the cooperative lot sizing problem under the capacity constraint. In order to find a stable and fair cost allocation, they propose solution approaches based on mathematical modeling. The proposed method is a constraint generation based algorithm, which is similar to the algorithm used to calculate the approximate nucleolus in the literature. Although these algorithms offer effective solutions for this kind of problem, they are not numerically efficient algorithms due to their enumerative nature, and hence they may not scale up for large-scale problems. The authors also claim that the proposed algorithm can handle the case with backorders even though they do not consider backorders explicitly in that study.

2.3 Machine Learning Methods

Improvements in machine learning algorithms are usually implemented for single target problems. Then, its improvements are applied to multi-target problems. The implementations of the algorithms for multi-target problems with high dependency among targets will be followed. The impact of high dependency among targets will be explained further. In the literature review below, similar improvement steps will be observed for the algorithms which will be applied in the thesis.

2.3.1 XGBoost with Chained Regression

2.3.1.1 Tree boosting algorithms and XGBoost

Tree boosting algorithms are ensemble methods that apply weak learners, which are decision trees, to reach a stronger learner by learning from errors with each new tree to improve model results [13]. The performance of tree boosting algorithms is dependent on the data and the weak learner [13]. Insufficient data, overly complex weak hypotheses, and noise in data can lead to failures to perform well [13]. On the other hand, tree boosting algorithms can perform significantly good results in extensive experiments on a wide range of problems with sufficient and well-constructed data [14]. The potential of tree-boosting algorithms makes them a possible best algorithm for the problem among the candidates.

XGBoost (Extreme Gradient Boosting) is an open-source machine learning system for tree boosting in predictive regression or classification problems. A machine learning competition site, Kaggle published that 17 of 25 top-3 challenge winning solutions for all competitions during 2015 applied XGBoost as the algorithm [14]. XGBoost's success is recognized in the machine learning community. Besides high predictive capability, shorter execution speeds, thanks to the multi-threaded approach of CPU cores, make XGBoost more popular for larger and more complex datasets [15]. This popularity makes XGBoost a usual benchmark for model comparisons in classification and regression problems. XGBoost will be reviewed as the first candidate as a tree boosting algorithm to compare with the

optimization results.

2.3.1.2 Multi-label predictions and chain transformation

XGBoost is mainly developed for single-label predictions. For multi-label problems, objectives are considered independent. For independent objectives, the value of each objective does not relate to the other objectives' values. However, in our case, each objective has a dependency on the other objectives with certain constraints. To reflect objective dependencies, model predictions for each objective should communicate with each other.

In multi-label classification literature, label correlations (i.e., dependencies) are crucial to consider for multi-label classification problems [16]. Modeling by taking the label correlations into the account improves the predictions. For this reason, a technique is needed to apply to the algorithms which do not take into the account the label dependency. Classifier chain transformation for multi-label classification problems is an approach to reflect label correlations with many advantages including reduced computational complexity and improved performance [16]. Chain transformation converts multi-dimensional dependency space into less complex problems. By providing logical ordering to the chain, predictive performance can be improved.

The multi-label classifier chain transformation method is also proposed to deal with multi-target regression problems with dependencies among targets [17]. Ensemble of regression chains (ERC) is one of the suggested approaches inspired by the classifier chains method. The idea is to develop different models for each target. According to the order of the developments, each output will be used in the next target's model development as an input. Based on problem subjects and relationships among targets, the order of chains can be random or ordered specifically. If there is a meaningful order among targets, deciding the order manually can be more beneficial to improving predictions. Otherwise, different randomness structures can be set to decide the order of model chains. For each target, a separate XGBoost regression model is developed in this study. After the first model

in the chain, the following models use previous models' outputs as inputs in their models besides the inputs from the dataset. Therefore, the following models carry multi-target dependencies from previous models to their models. The predictive performance of ERC can vary according to selected chain ordering. For this reason, different ordering combinations may return significantly different predictions.

2.3.1.3 Hyperparameter tuning

Tree boosting algorithms need hyperparameter tuning for optimal performance. The results vary according to applied hyperparameters. Finding a good hyperparameter set make the difference between the good and the bad models [18]. XGBoost also has several hyperparameters which define the structure of trees applied in models such as the number of estimators (i.e., trees), maximum depth of each tree, and learning rate [19]. Grid search is a popular method to get optimal hyperparameter set by searching through a manually defined subset of hyperparameters for the concerned machine learning algorithm [20]. However, finding optimal values may not be time-efficient empirically. Grid search has substantial computational costs depending on the dimension of the hyperparameter subset, which rises exponentially with the number of hyperparameters because each combination requires the training of a new model [20].

Bergstra and Bengio [21] proposed that randomly chosen trials from hyperparameter space are more efficient than grid search, especially for larger datasets and hyperparameter value sets. An XGBoost model can return performant predictions if enough iterations are applied for a random search. It is difficult to reach global optimum hyperparameter combinations with a random search. However, computational power and global optimum trade-off can find an adequate hyperparameter set with a random search. Therefore, random search is applied for the hyperparameter optimization stage for each single model in regression chains.

2.3.2 Multi-target Artificial Neural Networks

2.3.2.1 Artificial Neural Networks

Artificial Neural Networks (i.e., ANN) is a popular approach in artificial intelligence for classification, clustering, pattern recognition, and prediction [22]. ANN tries to learn data patterns by using nodes as neurons in a biological human brain. A node in an ANN model processes information to the next nodes in the structure by applying a simple mathematical function, which tries to mimic the learning process in biology. A significant difference between ANN and XGBoost models is that ANN models have a non-parametric environment [23]. This helps to develop models without prior knowledge about the distribution of data. The structure of the ANN models is important to find higher predictive performance.

The simplest form of ANN is shown in the Figure 1 [24]. There are three layers of ANN structure:

- Input Layer,
- Hidden Layer,
- Output Layer.

Each layer has nodes to transform information. Each node in the input layer represents an input variable to use in the model. The hidden layer connects to nodes in the input layer. More hidden layers and nodes increase the complexity of the model. The complexity of hidden layers helps to identify patterns in data to better predict multi-label outputs. However, a complex layer structure may cause overfitting. It is important to find a decent hidden layer structure to get better results in an ANN model. In the simplest form of ANN, the hidden layer connects to the one-node output layer, which is the output of the model. In our case, we have a multi-label regression problem. The number of nodes in the output layer will be equal to the number of the prediction labels.

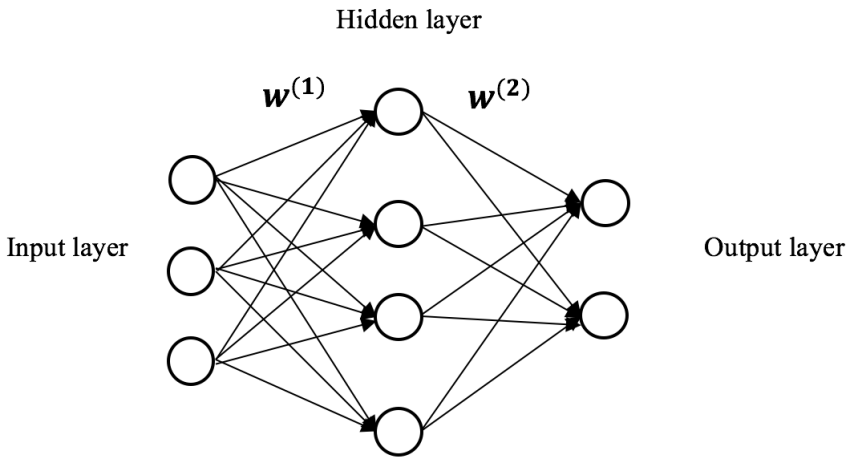

Figure 1: Simple artificial neural network (ANN)

A feedforward neural network (FFNN) is a machine learning algorithm in which each node in a layer connects to all the other nodes in the following layer [22]. Each connection has a weight, which represents the amount of information to carry from the current node to the next layers. The process continues in the forward direction from the input to the output by using the hidden layers. The information is not carried back to previous nodes in a feedforward neural network. Single-layer perceptron is the simplest FFNN technique. In a single-layer perceptron, layers are limited to input and output layers [25]. However, in a multilayer perceptron, there is a mapping between input and output vectors with hidden layers, which is different than a single-layer perceptron. Hidden layers in a multilayer perceptron can help to learn different data patterns.

The nodes of each hidden layer have an activation function, which is a mapping of summed weighted input to the next nodes [26]. Rectified Linear Unit (i.e., ReLU) is one of the most popular activation functions. ReLU speeds up learning convergence in the neural network [27]. ReLU is a piecewise linear function that returns 0 if the input is less than 0, otherwise, it returns the input directly (Equation 1) [28].

$$g(z) = max(0, z) \tag{1}$$

Goodfellow et al.[28] state that ReLU is linear for positive inputs, which makes models easier to optimize with gradient-based methods. This provides an opportunity to try different shapes of hidden layer structures in a given period. More hidden layers or more nodes in a hidden layer can be applied to compare the predictive performance of the different models.

To benefit from mentioned characteristics of ANN, Keras library will be applied [29]. Keras is an open-sourced deep learning library written in Python to develop ANN models. Keras is an easy-to-use and fast library with an application programming interface (i.e., API), which helps to try several structures in a short time. There are two main APIs in Keras:

- the sequential API,
- the functional API.

In the next sections, the differences between the sequential and the functional API will be explained.

2.3.2.2 Artificial Neural Networks with Sequential API

The sequential API in Keras allows the development of layer-by-layer models for multilayer network with the linear stack of layers [30]. A sequential model is created by adding layers that can have their activation functions.

ANN natively supports multi-label regression problems. ANN works well if the relationships between variables need to be understood [31]. In our case, a strong dependency is expected due to the nature of the problem. For example, an increase in variable cost on Day 1 impacts Allocation on Day 2 because it would be more beneficial to postpone a part of allocation from Day 1 to Day 2. ANN with multi-layer network can be a potential alternative for the solution. Multilayer network models will be developed with the sequential API in Keras. The sequential API will use all inputs to predict all the outputs. It is a simple way to apply ANN for prediction. The results will represent the strength of ANN to predict the multi-targets with high dependency even in the simplest form.

2.3.2.3 Artificial Neural Networks with Functional API

The functional API is another approach provided by Keras. It supports a more flexible hidden layer structure compared to the sequential API. The functional API offers more control over the layers compared to the sequential API. Multiple inputs can be part of different outputs' predictions. Therefore, the inputs will be applied only to the outputs with a certain level of relevancy.

Different hidden layers can be built for each output in a multi-label output regression problem [30]. Each input can be mapped to hidden layers separately. The information, which is not related to an output from the multi-output problem, can be ignored for this output directly at the beginning. In other words, the functional API allows fine-grained control of the structure of each layer and node. All inputs in the hidden layers can impact the predictions of all outputs in the sequential API. Input sets can be dedicated to the specific outputs with the functional API. The main advantage of this approach instead of creating separate models is to build joint models to predict multi-labels at the same time [30]. The multi-labels are dependent on each other due to the constraint function, so the model performance is expected to be improved with the functional API because the functional API allows for the creation of custom layer structures with dedicated input sets.

The data has information from different days as an input set from Day 1 to Day 10. Information from day 10 may be irrelevant to the output from day 1. For this reason, using inputs from day 10 to predict Allocation on day 1 may create noise in the model. A well-designed model with the functional API can keep dependencies among outputs without creating such input noise. Also, it may help to reach better results without computing unrelated inputs for each output. In the model, different day-range will be specified to get only relevant input data. In the sequential API, only one dataset is applied as the input with the data from day 1 to day 10. However, in the functional API, 10 different input datasets are created. Each dataset has different days as input, which is pre-defined as day range and targets one-day allocation. These day ranges will be used as the input

layer for each output of a given day. After certain hidden layer combinations, outputs will be combined in further layers to keep dependencies among outputs.

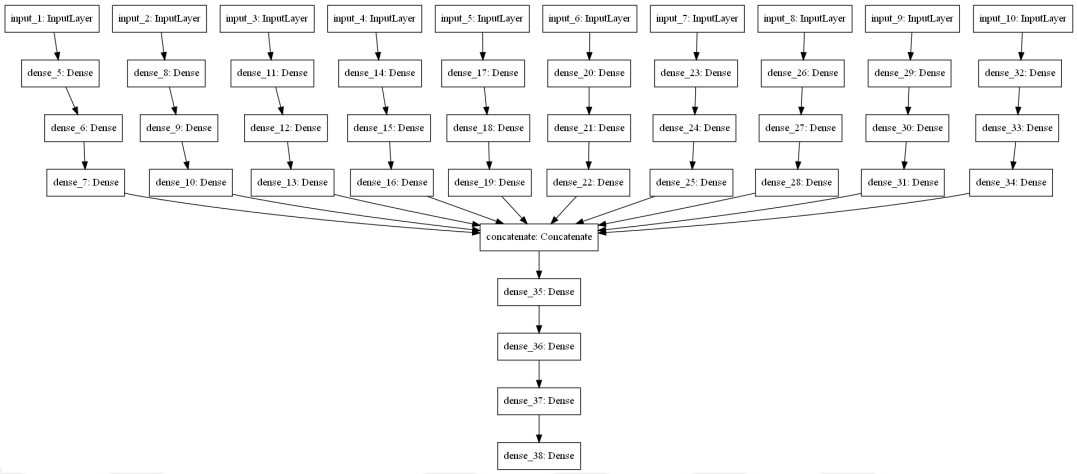

Figure 2: An example for the functional API neural networks structure

Figure 2 shows the example layer structure with the functional API. In the example, the model has 10 days of Allocation to predict. For example, if the day range is set to 2, Input_3 is an input layer and has information only from day 1 to day 5. This input dataset does not have any information from day 6 onwards. For this reason, dense_11, 12 and 13 hidden layers will only use data from day 1 to 5 and predict Allocation on day 3. After dense_13, predictions will be combined with the other days and continue to dense_35, 36, 37, and 38 to learn dependencies among targets. At the last layer, the model returns 10-day outputs as predictions. The predictions are expected to have higher accuracy in the functional API compared to the sequential API. Each model will be predicted with its relevant inputs in the first layers and the dependency among the targets will be learned in the latest layers. For this reason, the model will process more relevant information at each step.

CHAPTER III

COMPUTATIONAL STUDY

3.1 Datasets and Experimental Setup

3.1.1 Data

For the experiments, two datasets are created with and without the Inventory Capacity column to see the benefits of the information from the capacity at day 0. All model alternatives are applied to both datasets. Both datasets are created 10 days range data with 35000 different observations (Table 1).

Table 1: Basic information of Datasets

Dataset Name	Number of Rows	Number of Inputs	Total Number of Target Days	Description
Dataset 1	35000	61	10	Data without information from Day 0
Dataset 2	35000	62	10	Data with Inventory Capacity information from Day 0

In the Table 2, the definitions of all columns can be examined. Dataset 1 has 61 potential input columns. Dataset 2 has the same columns and the Inventory Capacity column as an extra. Both datasets have 10 target columns (Allocation). The decision of distribution of production amounts for T number of days in a production line is a computationally costly optimization problem. Cost is a constraint for the multi-number days in the optimization problem. The complexity of the problem increases as the number of days to predict increases. Each day has information about certain areas. These are Demand, Fixed Cost, Variable Cost, Holding Cost, Backorder Cost, and Capacity for the given number of days. In addition, Inventory Capacity is an input to reflect the capacity condition only at

day 0. So, for T number of days, we have $6 \times T+2$ number of input variables and T Allocation variables as output, where plus 2 indicates Cost and Inventory Capacity. As the number of days goes up, the complexity of the problem increases because new relations appear between inputs from different days with outputs.

Table 2: Column Definitions for total T number of days prediction

Column Name	Column Description
Allocation$t$	Number of allocation for each day t (target variable)
Cost	Cost as the constraint in total T days
Demand$t$	Demand for production for each day t
FixedCost$t$	Fixed cost spent even for no production for each day t
VariableCost$t$	Variable cost for each unit in production for each day t
HoldingCost$t$	Storage cost of each unit for each day t
BackorderCost$t$	Cost of delay in production of order for each day t
Capacity$t$	Capacity to keep in storage for each day t
InvCapacity	Inventory capacity condition at day 0 (the 2nd dataset)

3.1.2 Mean Squared Error

Mean squared error (i.e., MSE) is one of the most commonly used loss functions [32]. It is a popular method to try in all kinds of regression problems. Regression models try to minimize MSE. MSE is the mean of the squared differences between true and predicted values (Equation 2).

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{2}$$

where y is the true value and $\hat{y}$ is the predicted value.

MSE is sensitive to outlier values and it is important to penalize outliers more. The target values in the optimization have normal or normal-like distributions (e.g., Figure 3). For normal distributions, it is beneficial to penalize larger errors

more compared to smaller errors because the optimal prediction will be close to the mean. In the XGBoost model, MSE will be applied as the objective function to minimize the error.

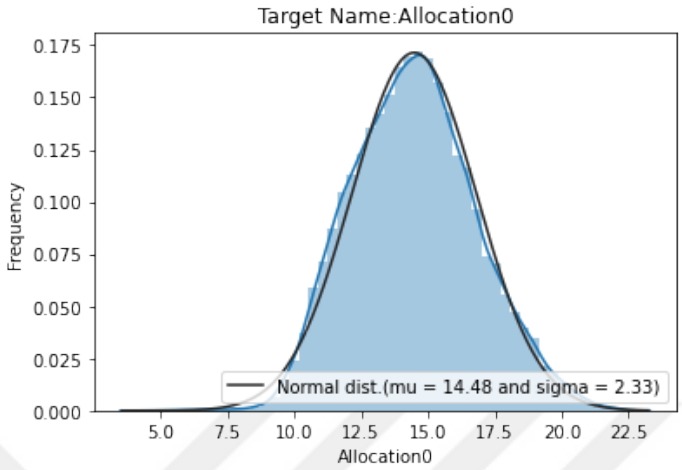

Figure 3: Normal distribution example for Allocation0 in Dataset 2

3.1.3 Data Exploration

The optimization problem is to predict Allocation1 to Allocation T, where T denotes the total number of days, with a given input set with the constraint in the Equation 3.

$$Cost = \sum_{t=0}^{T-1} Allocation_t \times Demand_t \tag{3}$$

Allocation denotes the Allocation amount in a given day and Demand denotes Demand input in a given day.

In addition to reaching high accuracy predictions by optimizing the evaluation metrics, the equation for the constraint should be checked. In the constraint, cost value and demand values from day 1 to day T are given. Allocation prediction sets must take the constraint into account while optimizing the predictions.

The target variables (i.e., Allocation), Holding Cost, and Backorder Cost are float values. The rest of the data is integer values. The characteristics of Allocation values differ in datasets 1 and 2. Allocation values in the dataset 2 are more spread with lower minimum (i.e., min), and higher maximum (i.e., max) and standard

deviation (i.e., std) values in the Table 4 compared to the same statistics for dataset 1 in the Table 3. This difference is the impact of Inventory Capacity in dataset 2. Thanks to Inventory Capacity, the optimization model can reach higher allocation numbers even at the same Cost level.

Table 3: Descriptive statistics for Allocation values in Dataset 1

Target Day	count	mean	std	min	25%	50%	75%	max
Allocation0	35000	13.35	2.01	8.67	11.82	13.46	14.85	19.69
Allocation1	35000	12.01	1.42	8.63	10.97	11.94	12.98	17.57
Allocation2	35000	12.14	1.41	8.53	11.13	12.14	13.13	17.02
Allocation3	35000	12.11	1.37	8.62	11.1	12.11	13.09	16.91
Allocation4	35000	12.1	1.37	8.61	11.11	12.1	13.08	17.05
Allocation5	35000	12.1	1.35	8.55	11.12	12.1	13.06	17.47
Allocation6	35000	12.14	1.37	8.56	11.15	12.16	13.12	17.28
Allocation7	35000	12.1	1.39	8.25	11.07	12.1	13.1	17.19
Allocation8	35000	11.88	1.27	7.68	10.99	11.85	12.76	16.68
Allocation9	35000	12.83	1.26	8.73	11.95	12.82	13.71	17.38

Table 4: Descriptive statistics for Allocation values in Dataset 2

Target Day	count	mean	std	min	25%	50%	75%	max
Allocation0	35000	14.48	2.33	4.42	12.81	14.44	16.01	22.37
Allocation1	35000	10.91	2.2	4.87	9.19	10.84	12.44	21.06
Allocation2	35000	12.88	1.82	3.95	11.63	12.79	14.07	22.31
Allocation3	35000	11.91	1.82	4.1	10.65	11.88	13.11	21.19
Allocation4	35000	12.33	1.79	3.21	11.1	12.26	13.49	22.88
Allocation5	35000	12.45	1.78	5.84	11.24	12.38	13.59	21.88
Allocation6	35000	11.78	1.8	3.29	10.55	11.74	12.95	21.16
Allocation7	35000	13.0	1.77	6.51	11.76	12.89	14.11	21.85
Allocation8	35000	10.65	2.02	1.47	9.22	10.62	11.99	21.31
Allocation9	35000	14.17	2.06	6.68	12.67	13.99	15.49	24.05

Due to the constraint of Cost, Allocation and Demand values for the same day are the counterparts. A lower Demand value for a given day is expected to lead to a higher Allocation value to check the equation for the Cost. It is less likely that both Demand and Allocation have higher values for a given Cost. For this reason, it is expected to observe a negative correlation between Demand and Allocation for a given day. However, in dataset 1, the expected relationship in terms of the correlation is limited (Table 5). In dataset 2, the negative Pearson correlations between Allocation and Demand are more significant for some days (Table 6). This difference increases the expectation for better predictive performance in dataset 2.

Table 5: Correlations of Allocation and Demand for the same day in Dataset 1

Target Day	Pearson Correlation
Day 0	-0.14
Day 1	0.02
Day 2	-0.04
Day 3	-0.04
Day 4	-0.04
Day 5	-0.04
Day 6	-0.05
Day 7	-0.02
Day 8	-0.01
Day 9	-0.16

Table 6: Correlations of Allocation and Demand for the same day in Dataset 2

Target Day	Pearson Correlation
Day 0	-0.49
Day 1	0.05
Day 2	-0.34
Day 3	-0.16
Day 4	-0.25
Day 5	-0.26
Day 6	-0.15
Day 7	-0.36
Day 8	0.05
Day 9	-0.55

Moreover, Pearson correlation values for day t and T - t ($T = 9$ in this example) have similar values. They show some symmetry-like behaviors. They are more observable in dataset 2. For both datasets, the lowest Pearson correlations are in the first and the last day (Figure 4 and Figure 5). The highest Pearson correlations are in the first second and the last second days. This may indicate that the optimization solution tends to fit the outlier days more in terms of the constraint. In the following sections, a proposed model that reflects a more comprehensive understanding of the constraint might find better results compared to the optimization solution.

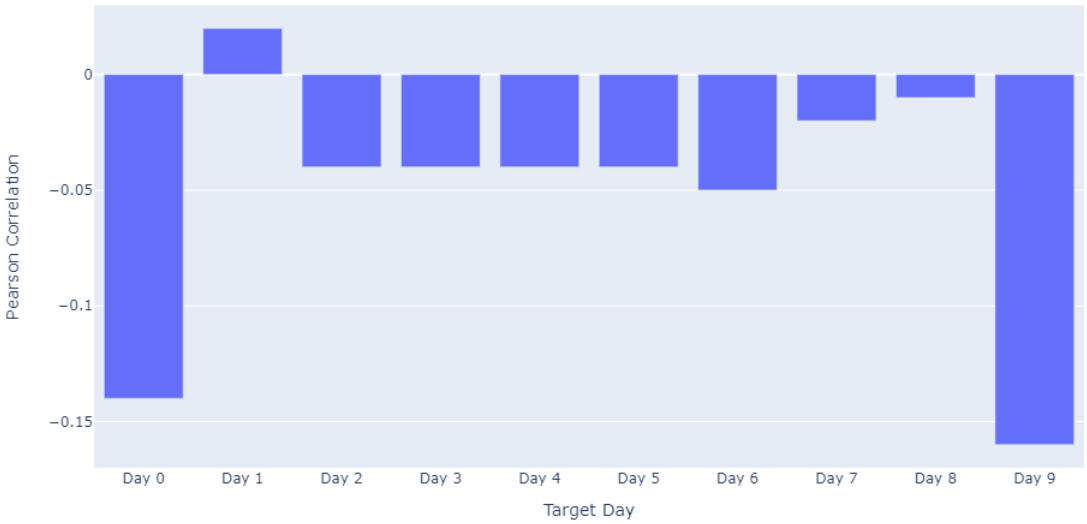

Figure 4: Correlations of Demand and Allocation in a given day for Dataset 1

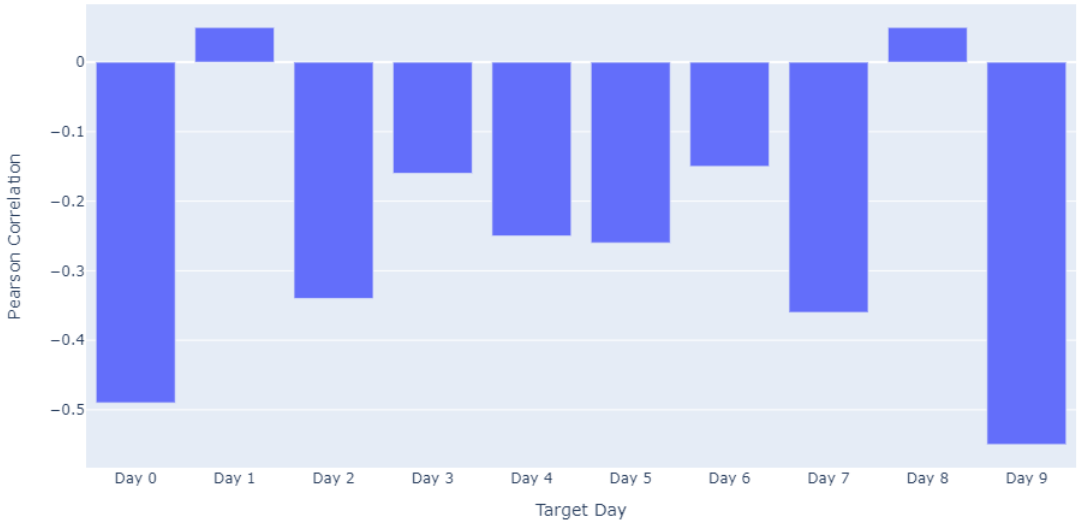

Figure 5: Correlations of Demand and Allocation in a given day for Dataset 2

Before the modeling phases, both datasets 1 and 2 are split into 67%-33% training-test datasets. The training dataset will be used for the model training. Then, the predictive performance of the models will be evaluated on the test datasets.

The thesis aims to solve the problem with machine learning techniques such as XGBoost and artificial neural networks. 3 alternative solutions to the optimization technique as follows:

- Chained Regression: XGBoost,
- Artificial Neural Network with the sequential API,
- Artificial Neural Network with the functional API.

3.2 Results of XGBoost with Chained Regression

XGBoost is a tree boosting algorithm, which will be the first solution candidate for the optimization problem. For modeling purposes, the following steps will be applied:

- create hyperparameter dictionary,

- create a machine learning pipeline with regression chains,
- apply XGBoost to the pipeline,
- optimize hyperparameters regarding the evaluation metrics,
- apply a transformation to satisfy the equation.

3.2.1 Hyperparameter Dictionary

XGBoost has several hyperparameters which directly impact the performance of the model. A hyperparameter dictionary is created to optimize hyperparameters for better performance. Finding the optimal hyperparameter subset can significantly improve the predictive performance of the model, which had been proved theoretically and empirically [19]. The main disadvantage of searching the optimal subset is that the process is time-consuming. For this reason, hyperparameter set should be carefully selected. A random search with many iterations will try to find the highest predictive performance among the iterations to set a reasonable time limit for this step. Hyperparameters that will be optimized for the model are in the Table 7 with the definitions from the XGBoost document [33].

Table 7: Hyperparameter definitions in XGBoost

Hyperparameter	Description
min_child_weight	Minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.
learning_rate (eta)	Regularization parameter to shrink feature weights in each boosting step. Shrinkage of feature weights makes the boosting process more conservative.
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.
subsample	Subsample ratio of the training instances for every boosting iteration.
colsample_bytree	The subsample ratio of columns when constructing every tree.
colsample_bylevel	The subsample ratio of columns for every level.
max_depth	Maximum depth for each tree can grow. Increasing values may cause overfitting.
n_estimators	Number of decision trees to be boosted. The lower values may lead to underfitting.

3.2.2 Machine Learning Pipeline

A machine learning pipeline is a workflow to optimize all modeling steps in an execution [34]. A machine learning pipeline is created with regression chains to simplify the iterations. Each prediction step in the pipeline is designed to use the prediction as an input point for the following step in the regression chains. Each model will have the input dataset and previous models' predictions as a new input dataset (Figure 6).

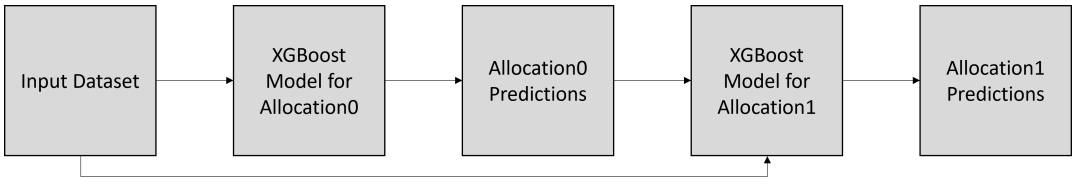

Figure 6: Example for 2-days Regression Chains structure

3.2.3 XGBoost and Hyperparameter Optimization

As mentioned earlier, mean squared error (i.e., MSE) will be applied as the objective function in XGBoost. For each target (10 targets in this experiment), a different XGBoost model will be developed with a given hyperparameter subset to minimize MSE. The hyperparameters will be optimized to minimize the error function MSE of the regression chains with random search iterations. Random search will be iterated for hundreds of different hyperparameter subsets.

The order of chains might bring different results related to dependencies among the targets. The time-wise order from Day 1 to Day 10 and the random order are applied for different iterations. The motivation to try random ordering is that calculating different points in time range may bring valuable information about data patterns that we cannot find with the time-wise order. 4-fold cross-validation is applied with different hyperparameter subsets and regression chains structure to find the best settings. The lowest MSE performance of the pipeline among different hyperparameter subsets is selected for the further steps.

3.2.4 Output Transformation

Output transformation is required to satisfy the constraint equation of Cost. An output transformation function is introduced to apply after the prediction step. Cost is the constraint (Equation 4) for the optimization problem.

$$C = \sum_{t=0}^{T-1} A_t \times D_t \tag{4}$$

where C denotes Cost input, A_t denotes the Allocation amount for the day t and D_t denotes Demand input for the day t.

The same equation for Allocation predictions is also introduced (Equation 5).

$$\hat{C} = \sum_{t=0}^{T-1} \hat{A}_t \times D_t \tag{5}$$

where $\hat{C}$ is the result of the equation which reflects the predicted Cost regarding to predicted Allocation, $\hat{A}_t$ is the predictions for the Allocations for the day t

and D_t denotes Demand input for the day t. $\hat{C}$ must be equal to C to satisfy the constraint. However, the model predictions can have different values which causes not satisfying the constraint. To ensure to satisfy the constraint, the output transformation is introduced (Equation 6).

$$\hat{A}_{transformed\_t} = \hat{A}_t \times \frac{C}{\hat{C}} \tag{6}$$

where $\hat{A}_{transformed\_t}$ is the transformed predictions for the day t which satisfies the constraint, $\hat{A}_t$ is the predictions from the model for Allocations for the day t and D denotes Demand input for the day t. $\hat{A}_{transformed\_t}$ values satisfy the constraint. For this reason, the predicted values from the selected model are transformed with given transformation function. This transformation step decreases the errors in the predictions and makes sure the constraint is satisfied.

3.2.5 Result

The best models in terms of the lowest MSE are applied to the stability scoring method, which is developed for the optimization problem to compare the model results with the optimization results. A lower stability score means a more stable and better model result.

As mentioned earlier, two different datasets are created for modeling purposes. The first dataset is without the Inventory Capacity column, the second dataset is with it. The difference in the model performances for two different datasets may help to understand the contribution of the Inventory Capacity to the models.

For the first dataset, which is without the Inventory Capacity information, the stability scoring of the optimization result is 3.16%. The XGBoost alternatives perform worse with 5.73%. This means that XGBoost is unable to learn the dependency among days as well as the optimization result. However, with the introduction of the Inventory Capacity, the optimization result performs 17.48% while XGBoost performs 14.94% in terms of stability score in dataset 2. The Inventory Capacity provides a better understanding of dependencies among the information

from different days and helps to decrease the dependency among Allocations on different days for better XGBoost results.

3.3 Results of Neural Networks with Sequential API

The second candidate for the optimization solution is the sequential neural networks approach. Sequential neural networks are the basic artificial neural networks API by Keras. The following steps are applied to develop models on neural networks with the sequential API:

- create hidden layer structure,
- feature transformation,
- neural networks training,
- output transformation.

3.3.1 Hidden Layers

The sequential API allows setting the number of nodes for each hidden layer. For different iterations, different networks are created with different hidden layer combinations to balance fitting. The output layer has 10 nodes for the 10-Day estimation problem, which is one for each target estimation. For example, in Figure 7, 62 inputs are applied to the model in the input layer. Then, it follows 4 hidden layers with 120, 80, 60, and 30 outputs, respectively. In the output layer, it returns 10 outputs as the Allocation estimations for each day.

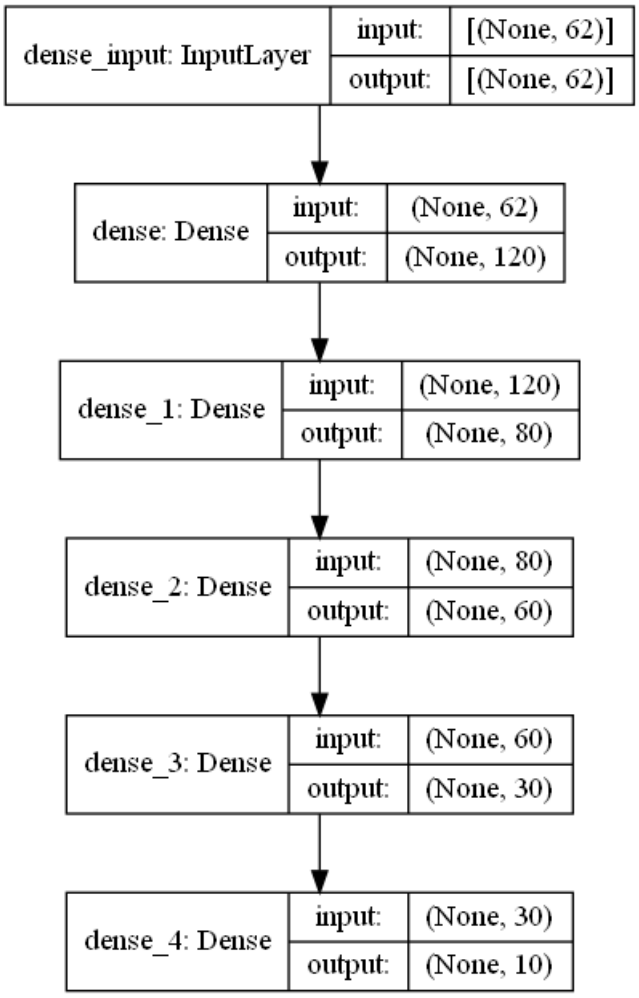

Figure 7: An example for the sequential API neural networks structure

3.3.2 Feauture Transformation

Data preprocessing has the potential to improve the predictive performance of the modeling algorithm. Standardization is a technique to transform the dataset features to scale around a certain value or range [35]. Large ranges in a feature can impact the evaluation metric dramatically. The standardization process can decrease this impact and lead to higher performance. For the neural networks modeling, MinMaxScaler from the Sklearn library is applied to re-scale each input between 0 and 1 [36]. The scaler calculates each input column's minimum and maximum values and applies Equation 7.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{7}$$

where X denotes the input column, X_{scaled} denotes the scaled input column, X_{min} denotes the minimum value of the input column and X_{max} denotes the maximum value of the input column. X_{scaled} is calculated for each input column and used in the model instead of the original column values.

3.3.3 Neural Networks

Neural networks with the sequential API have some points to be decided before modeling. ReLU is applied as the activation and MSE is applied as the loss function. For the initial random weights of the layers, an initializer should be set. He uniform initializer is applied for the weights of the layers. He uniform initializer works better with ReLU activation function [37]. He uniform initializer uses samples from a uniform distribution between +/- limit value where the limit is in the Equation 8.

$$limit = \sqrt{\frac{6}{inp}} \tag{8}$$

where inp is the number of input units in the weight tensor.

For the optimizer parameter, Adam optimization is selected. Adam optimization is a stochastic gradient descent method that is based on the adaptive estimation of first-order and second-order moments. Adam optimization is computationally efficient with lower memory requirement, invariant to diagonal rescaling of gradients, and is efficient for large datasets and parameters [38]. Neural networks with the sequential API are trained in several iterations with different hidden layer structures and given parameters.

3.3.4 Output Transformation

For neural networks with the sequential API, the same output transformation mentioned in the XGBoost section is applied (Equation 6). Additionally for neural networks with the sequential API predictions, the output transformation improved the predictions compared to the predictions before the transformation.

3.3.5 Result

The best results in the test dataset in terms of the lowest MSE are selected for stability scoring to compare with the optimization solution and the other offered models. For the first dataset, which is without Inventory Capacity information, the stability scoring of the optimization result is 3.16%. The best sequential neural networks model performs worse than the optimization results with a 4.71% stability score. Similar to the XGBoost result, with the introduction of Inventory Capacity, the optimization result performs 17.48% while the best sequential neural networks model performs 15.53% in terms of stability score. It performs worse than the XGBoost result but still, better than the optimization result. The Inventory Capacity feature still plays an important role to understand target dependencies in the dataset.

3.4 Results of Neural Networks with Functional API

The final alternative for the optimization solution is the functional API neural networks approach. The functional API neural networks is a customizable API by Keras. The inputs can be mapped through pre-defined nodes and layers can be adjusted accordingly. Therefore, the dependencies among the targets can be predicted better by using the dependencies among the inputs. To predict allocation on day 1, the sequential method uses all inputs even from day 10. However, a day range can be set for each target with the functional API. For example, inputs from day 1 to 3 can be applied for the prediction of allocation on day 1. So, the model can be more focused to learn the dependencies for the given days. The functional API follows similar modeling steps to the sequential API. A customization step will be added while creating the hidden layer structure. The following steps are applied to develop models on neural networks with the functional API:

- create hidden layer structure,
- feature transformation,

- neural networks training,
- output transformation.

3.4.1 Hidden Layers

For the 10 days estimation problem, 10 different input datasets are created. Each input dataset has the information from 2 days before to 2 days later. For example, for the day 7 allocation, the input dataset has the information from day 5 to day 9 (Figure 8). For the first 2 and the last 2 days, the information and the input dataset are limited because they have less number of days in their scope.

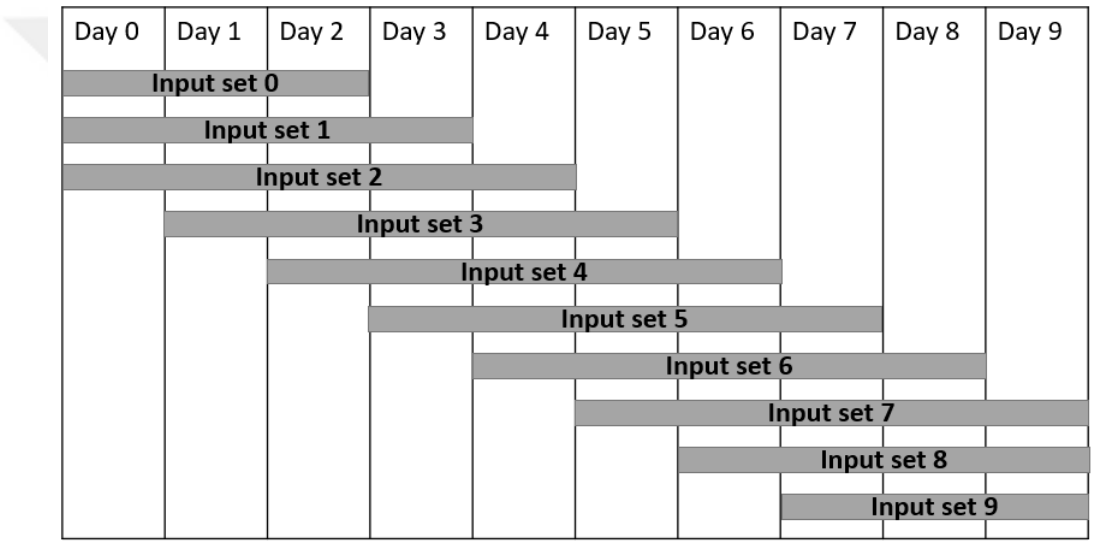

Figure 8: An example for 10 days inputs datasets in the functional API models

Each input dataset is applied to an input layer. Each input layer follows 3 hidden layers with 40 nodes in each layer independent from the other input datasets and the neural networks. Then, the estimations are used as the inputs for the further hidden layers. After three more hidden layers with 20 nodes in each, the output layer returns 10 output nodes for 10 days of estimation. Each node represents a one-day prediction in terms of Allocation.

3.4.2 Feauture Transformation

The same technique is applied for the feature transformation step in the functional API model with the sequential API model. MinMaxScaler scales the input values

between 0 and 1. This approach is also beneficial in the functional API model.

3.4.3 Neural Networks

Neural networks with the functional API have the same parameters to decide before modeling as the sequential API. For these decisions, the same functions are applied. The activation function is ReLU. The loss function is MSE the same as the sequential API. He uniform initializer is also applied in the functional API. The number of inputs is different in the functional API due to the customization of the nodes to 10 different flows. For this reason, the limits for the He uniform initializer are wider. Adam optimization is also applied as the optimizer parameter. Neural networks with the functional API are trained in several iterations with different hidden layer structures and given parameters.

3.4.4 Output Transformation

The output transformation technique remains the same for neural networks with the functional API with previous alternatives (Equation 6). The output transformation is required to check the constraint with the model predictions. For neural networks with the functional API predictions, the output transformation improved the predictions compared to the predictions before the transformation.

3.4.5 Result

Neural networks with the functional API approach improved the performance of the models compared to the sequential neural networks models. For the first dataset, which is without the Inventory Capacity variable, the best the functional API model scored 4.57% stability score, which was 3.16% for the optimization, 5.73% for the XGBoost, and 4.71% for the sequential neural network model. The functional API neural networks performed better than the other alternatives, but it still scored worse than the optimization solution.

For the second dataset, which includes the Inventory Capacity variable, the

functional API model scored 15.45%, which was 17.48% for the optimization solution, 14.94% for the XGBoost, and 15.53% for the sequential neural network model. The functional API performs better than the optimization solution and the sequential neural network model, but the XGBoost model performed better than the functional API model.

3.5 Results

As mentioned in the previous sections, each model performs differently in terms of the stability score with and without the Inventory Capacity variable (Table 8). For dataset 1, the alternatives cannot overperform the optimization solution. Both XGBoost and the neural networks solutions cannot learn the dependencies among the targets as well as the optimization solution. However, the offered solutions outperform in terms of the stability score in dataset 2. Inventory Capacity helps to improve the performance of the offered models. Its improvement has different effects for each model. For the XGBoost result, its impact is much higher than both the sequential and the functional API neural networks models. Regression Chains are successful to learn target dependencies with the help of the Inventory Capacity column.

Table 8: Experiment results based on different models and datasets

Models	*Dataset* 1	*Dataset* 2
Optimization Solution	3.16%	17.49%
XGBoost	5.74%	14.95%
Sequential API	4.72%	15.54 %
Functional API	4.58%	15.45%

For both datasets, the functional API model outperforms the sequential API model. Modeling each target in the separate neural networks and combining their outputs improved model performance.

CHAPTER IV

CONCLUSION

In this thesis, the problem of allocating the joint cost of a capacitated dynamic lot sizing problem among the companies receiving the product from a common supplier is considered. Under the capacity restriction, not only the underlying optimization problem but also the cost allocation problem become much more complex. We show that by using novel machine learning methods the core of the corresponding cooperative game might be approximated accurately. Based on a detailed computational study, we observe that our proposed mechanisms perform significantly better than the other proposed allocation mechanisms in almost all criteria under different problem settings with the existence of Inventory Capacity information.

REFERENCES

[1] M. H. Wagner and T. M. Whitin, "Dynamic version of the economic lot-sizing model," *Management Science*, vol. 5, no. 1, pp. 89–96, 1958.

[2] G. R. Bitran and H. H. Yanasse, "Computational complexity of the capacitated lot size problem," *Management Science*, vol. 28, no. 10, pp. 1174–1186, 1982.

[3] M. Florian and M. Klein, "Deterministic production planning with concave costs and capacity constraints," *Management Science*, vol. 18, no. 1, pp. 12–20, 1971.

[4] N. Brahimi, S. Dauzere-Peres, N. Najid, and A. Nordli, "Single item lot-sizing problems," *European Journal of Operational Research*, vol. 168, no. 1, pp. 1–16, 2006.

[5] L. Buschkuhl, S. H. F. Sahling, and H. Tempelmeier, "Dynamic capacitated lot-sizing problems: a classification and review of solution approaches," *OR Spectrum*, vol. 32, no. 2, pp. 231–261, 2010.

[6] N. Brahimi, N. Absi, S. Dauzere-Peres, and A. Nordli, "Single-item dynamic lot-sizing problems: an updated survey," *European Journal of Operational Research*, vol. 263, no. 1, pp. 838–863, 2017.

[7] V. den Heuvel, P. Borm, and H. Hamers, "Economic lot-sizing games," *European Journal of Operational Research*, vol. 176, no. 2, pp. 1117–1130, 2007.

[8] D. Xu and R. Yang, "A cost-sharing method for an economic lot-sizing game," *Operations Research Letters*, vol. 37, no. 1, pp. 107–110, 2009.

[9] M. Gopaladesikan, N. Uhan, and J. Zou, "A primal–dual algorithm for computing a cost allocation in the core of economic lot-sizing games," *Operations Research Letters*, vol. 40, no. 1, pp. 453–458, 2012.

[10] A. Toriello and N. Uhan, "Dynamic cost allocation for economic lot-sizing games," *Operations Research Letters*, vol. 42, no. 1, pp. 82–84, 2014.

[11] X. Chen and J. Zhang, "Duality approaches to economic lot-sizing games," *Production and Operations Management*, vol. 25, no. 7, pp. 1203–1215, 2016.

[12] J. Drechsel and A. Kimms, "Cooperative lot-sizing with transshipments and scarce capacities: solutions and fair cost allocations," *International Journal of Production Research*, vol. 49, no. 9, pp. 2643–2668, 2011.

[13] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[15] R. Santhanam, N. Uzir, S. Raman, and S. Banerjee, "Experimenting xgboost algorithm for prediction and classification of different datasets," *International Journal of Control Theory and Applications*, vol. 9, no. 40, 2016.

[16] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, no. 3, pp. 333–359, 2011.

[17] E. Spyromitros-Xioufis, G. Tsoumakas, and I. V. William Groves, "Multi-target regression via input space expansion: treating targets as inputs," *Machine Learning*, vol. 104, no. 1, pp. 55–98, 2016.

[18] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *NIPS'11: Proc. of the 24th International Conference on Neural Information Processing Systems*, pp. 2546–2554, 2011.

[19] S. Putatunda and K. Rama, "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of xgboost," *Proc. of the 2018 International Conference on Signal Processing and Machine Learning*, pp. 6–10, 2018.

[20] R. Schaer, H. Müller, and A. Depeursinge, "Optimized distributed hyper-parameter search and simulation for lung texture classification in ct using hadoop," *Journal of Imaging*, vol. 2, no. 2, pp. 55–98, 2016.

[21] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[22] O. I. Abiodun *et al.*, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, 2018.

[23] S. Walczak, "Artificial neural networks," in *Encyclopedia of Information Science and Technology, Fourth Edition*, pp. 120–131, IGI Global, 2018.

[24] *Going from logistic regression to single-layer neural networks.* oreilly.com/library/view/r-deep-learning/9781788478403/0c4ae722-74b3-422b-a67d-4b21e4aa1c96.xhtml, 2 May 2022.

[25] Šarūnas Raudys, "Evolution and generalization of a single neurone: I. single-layer perceptron as seven statistical classifiers," *Neural Networks*, vol. 11, no. 2, pp. 283–296, 1998.

[26] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)-a review of applications in the atmospheric sciences," *Atmospheric Environment*, vol. 32, no. 14, pp. 2627–2636, 1998.

[27] K. Hara, D. Saito, and H. Shouno, "Analysis of function of rectified linear unit used in deep learning," *International Joint Conference on Neural Networks (IJCNN)*, 2015.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[29] *Keras*. keras.io, 5 May 2022.

[30] F. Chollet, *Deep learning with Python*. Manning, 2018.

[31] V. S. Dave and K. Dutta, "Neural network based models for software effort estimation: A review," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 295–307, 2014.

[32] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[33] *XGBoost Parameters*. xgboost.readthedocs.io/en/stable/parameter.html, 17 March 2022.

[34] *What are Azure Machine Learning pipelines?* docs.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines, 14 February 2022.

[35] V. N. Raju, K. P. Lakshmi, V. M. Jain, A. Kalidindi, and V. Padma, "Study the influence of normalization/transformation process on the accuracy of supervised classification," *Proceedings of the 3rd International Conference on Smart Systems and Inventive Technology, ICSSIT*, pp. 729–735, 2020.

[36] *Sklearn MinMaxScaler*. scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html, 6 May 2022.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[38] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference for Learning Representations, San Diego*, 2015.

VITA

Furkan Kasapoğlu graduated from Bornova Anatolian High School in 2012. He received his bachelor's degree in Economics from Boğaziçi University in 2017. Since his graduation, he has been working for four years in ING as Data Scientist. He joined the Master of Science program in Data Science at Özyeğin University in 2020. Currently, Furkan is working as Data Scientist in Afiniti for more than a year. His research focuses on machine learning and deep learning.