A MIXED-INTEGER PROGRAMMING APPROACH TO EXAMPLE-DEPENDENT COST-SENSITIVE LEARNING

by

Tarkan Temizöz

B.S., Industrial Engineering, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2021

ACKNOWLEDGEMENTS

Above all, I would like to express my utmost gratitude to Assist. Prof. Mustafa Gökçe Baydoğan for his invaluable and continued guidance throughout this study. His endless patience and support taught me how to conduct a proper research, which I will bear in mind for the rest of my career. I hope our joint studies will continue in days to come. I am also grateful to the members of Department of Industrial Engineering, I learned a lot from them. Moreover, my sincere thanks go to Mert Yüksekgönül, who helped me in this work with his brilliant ideas and talent.

I would like to thank my parents Fatma, Cemal and my elder brother Tolga for their lifelong support. Their encouragement and love always help me to strive for the best. I also want show thankfulness to my other family members Yüksel, Güldane, Aysun and Utku.

My special thanks go to Gizem, Burçak and Mina. The words are not enough to describe how I feel lucky to have such amazing, cool and crazy friends.

My oldest buddies Çağrı and Fazılhan deserve a huge gratitude. I will always look forward to our future meetings. I also want to thank my former classmates and lifelong friends Ekin, Fethi and all other members of Kuzular.

I would like to acknowledge Data Science and Optimization Lab members Elif, Buğra, Selin and my fellow collegues Feyyaz, Çiğdem and Orkun for their friendship and many shared memories.

I would not miss expressing my gratitude for being a member of Boğaziçi Universitesi Denizcilik ve Yelken Kulübü, I enjoyed every single second when we sail together. I also owe Arda a special thank for being one of the closest during my graduate study.

Finally, I want to thank Scientific and Technological Research Council of Turkey (TUBITAK) for their financial support during my graduate study through BIDEB-2210A scholarship. This support was especially valuable in the early days of my graduate study.

ABSTRACT

A MIXED-INTEGER PROGRAMMING APPROACH TO EXAMPLE-DEPENDENT COST-SENSITIVE LEARNING

In this research, we study example-dependent cost-sensitive learning that brings about varying costs/returns based on the labeling decisions. Originating from decision-making models, these problems are distinguished in areas where cost/return information in data is focal, instead of the true labels. For example, in churn prediction and credit scoring, the primary aim is to build predictive models that minimize the misclassification error. Then, the outputs of the model are used to make decisions to minimize/maximize the costs/returns. In other words, prediction and decision making are considered as two separate tasks which may provide local optimal solutions. To resolve such problems, we propose a general strategy to incorporate instance-based costs/returns in a learning algorithm. Specifically, the learning problem is formulated as a mixed-integer program to maximize the total return. Given the high computational complexity of the mixed-integer linear programming problems, this model can be practically inefficient for training on large-scale data sets. To address this, we also propose *Cost-sensitive Logistic Regression*, a nonlinear approximation of the formulated linear model, which benefits from gradient descent based optimization. Our experimental results show that the proposed approaches provide better total returns compared to traditional learning approaches. Moreover, we show that the optimization performance of the mixed-integer programming solver can be enhanced by providing initial solutions from Cost-sensitive Logistic Regression to the mixed-integer programming model.

ÖZET

ÖRNEKLERE-BAĞLI MALİYET-DUYARLI ÖĞRENMEYE KARIŞIK TAMSAYI DOĞRUSAL PROGRAMLAMA YAKLAŞIMI

Bu araştırmada, etiketleme kararlarına göre değişen maliyetler/getiriler alan, örneklere-bağlı maliyet-duyarlı öğrenmeyi inceliyoruz. Karar verme modelleriyle ortaya çıkan bu tip problemler, veride doğru etiket bilgisinden çok maliyet/getiri bilgilerinin odaklandığı alanlarda ayırt edilir. Örneğin, müşteri kaybı tahmininde ve kredi puanlamasında birincil amaç, yanlış sınıflandırma hatasını en aza indiren tahmin modelleri oluşturmaktır. Daha sonra, modelin çıktıları maliyetleri/getirileri en aza indiren/maksimize eden kararlar almak için kullanılır. Diğer bir deyişle, tahmin etme ve karar verme, yerel optimal çözümlere yol açabilen iki ayrı yöntem olarak görülür. Bu tür sorunları çözmek için, örneklere-bağlı maliyetleri/getirileri bir öğrenme algoritmasına dahil etmek için genel bir strateji öneriyoruz. Spesifik olarak, öğrenme problemi toplam getirileri maksimuma çıkarmak amacıyla tamsayı karışık doğrusal programlama olarak formüle edilmiştir. Karışık tamsayı doğrusal programlama problemlerinin yüksek hesaplama karmaşıklığı göz önüne alındığında, bu model büyük ölçekli veri kümeleri üzerinde eğitim için pratikte verimsiz olabilir. Bunu ele almak için, gradyan azalma merkezli eniyilemeden fayda sağlayan ve formüle edilen doğrusal mo- dele doğrusal olmayan yaklaşım sunan Maliyet-Duyarlı Lojistik Regresyon modelini de ileri sürüyoruz. Deneysel sonuçlarımız, önerilen yaklaşımların geleneksel öğrenme yaklaşımlarına kıyasla daha iyi toplam getiri sağladığını göstermektedir. Ayrıca, Maliyet-Duyarlı Lojistik Regresyon modelinden alınan başlangıç çözümleri tamsayı karışık doğrusal program modeline verildiğinde, tamsayı karışık doğrusal program çözücüsünün eniyileme performansının geliştirilebilir olduğunu da gösteriyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C	Total number of classes in the data set
$d_j^{(i)}$	Binary variable showing whether the instance i is predicted as class j
$\mathbb{J}$	Set of all classes in the data set
$\mathbb{K}$	Set of all features of an instance in a data set
K	Total number of features of an instance in the data set
lb	Lower bound on the score values
M	A big number to be used in revised softmax function
N	Total number of instances in the data set
$p_j^{(i)}$	Continuous variable representing the score value of the class j for the instance i
$pr^{(i)}$	The profit the bank made from customer i
$o_j^{(i)}$	The outcome assigned to class j of instance i in the simulations
$q_{jt}^{(i)}$	Binary variable showing whether the score of class j is greater than the score of class t
$r_j^{(i)}$	The return of predicting instance i as class j
$\mathbb{S}$	Set of all instances in the data set
$\mathbb{T}_j$	Set of all classes greater than j in the data set
x	The data set
$y^{(i)}$	Binary or positive integer, shows the true label of the instance i
ub	Upper bound on the score values
ϵ	Margin between the score values corresponding to an instance
$\theta_k^{(j)}$	The coefficient of the class j for the feature k

LIST OF ACRONYMS/ABBREVIATIONS

BMR	Bayesian Minimum Risk
BNN	Binarized Neural Network
CSLR	Cost-sensitive Logistic Regression
DNN	Deep Neural Network
EDCS	Example-dependent Cost-sensitive
MIP	Mixed-integer programming model
MIP-WI	Mixed-integer programming with initial solutions model
ML	Machine Learning
NN	Neural Network
ReLU	Rectified Linear Unit
RF	Random Forest
RP	Random Patches

1. INTRODUCTION

In the context of machine learning, classification algorithms are used to predict the class information of test data, given a set of training instances of size N $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$, where $x^{(i)} \in \mathbb{R}^K$ is a feature vector and $y^{(i)} \in \{1, \ldots, C\}$ represents its corresponding class information. Standard methods aim at learning a mapping function f from input variables $x^{(i)}$ to discrete output variable $y^{(i)}$ that minimizes the misclassification error. Some classification problems, however, are unique in nature that these common techniques fall short of providing good results. Example-dependent cost-sensitive (EDCS) problems stand out as one of them, in which every instance has its own cost for misclassification or revenue for correct classification.

In fact, cost-sensitive learning is a subject in real-world scenarios. A similar challenge is faced in the example of credit scoring where banks evaluate the credit applicants to decide whether giving a loan to a particular customer would bring a positive profit for the creditor. For instance, a bank has a variable set of customers that poses clear segmentation from low to high risk. The amount of granted loans varies among the customers and they are offered on different interest rates so that the bank is also expected to get a varying amount of money back. Defaulting happens in two ways: The applicant either pays a partial dividend of his/her debt back, or there is no repayment at all. Hence, including the loan it grants, the bank gains different returns from each customer, a positive or a negative amount. Approving a customer for a credit is a critical decision for the bank because customer default means a significant loss for that bank. Hence, the aim of the classification model to detect the potential defaulters should become maximizing the total returns. Traditional accuracy-driven classification methods do not take instance-based returns into account; instead, learning is performed based on constant misclassification error. To maximize the profits and decide whether to grant a loan to a particular customer or not, a financial institution needs to develop a model that incorporates example-dependent costs. This introduces the concept of cost-sensitive learning with example-dependent costs [1].

The literature on cost-sensitive learning first appeared to solve the class imbalance problem and then became interested in class-dependent costs where the cost information is based on the classes. Example-dependent cost-sensitive learning; however, is interested in tasks with different costs among each instance, not just among classes [2]. Such problems can be in binary or multiclass nature and have known cost information. Thus, the process of introducing example-dependent costs is partitioned into three categories based on how example-dependent costs are fed into the model: prior training, during training or after training [3, 4].

In prior training, the operations are based on the data level, each instance is weighted according to its cost and the new training set is created thereupon by oversampling the instances with low misclassification costs [5] or undersampling (rejection sampling) the instances with high costs [6, 7]. The studies mostly focus on the class imbalance problem with binary classification and address the issue by oversampling the minority class and undersampling the majority class. However, these methods require more computational operations and demonstrate some downsides [8]. Undersampling brings about loss of information about the majority class and oversampling creates copies of minority class artificially which in turn may distort the data. In addition, studies show that prior training is only effective in binary classification case [7], and even have a negative effect in multiclass case [9].

Alternatively, incorporating example-dependent costs can be also done after training, as in Bayes Minimum Risk (BMR) method [10]. After learning a model that assigns probabilities to each class, BMR builds a decision model to measure the trade-off between classes by minimizing the expected loss (or maximizing the expected revenue) for each instance and classification takes place accordingly. This approach, on the other hand, is limited to binary classification.

Lastly, one can introduce example-dependent cost information when learning occurs. In such approaches, the parameters of the model are learned incorporating these varying costs and researches are based on modifying the learning phase of a machine

learning algorithm in a cost-sensitive manner. A deep analysis of the previous studies that introduce costs during training is presented in Chapter 3.1.

In this thesis, a novel approach is developed to analyse and evaluate example-dependent cost-sensitive learning. The classification problem is reformulated such that introducing the vector of returns $r^{(i)} \in \mathbb{R}^C$ corresponding to each class for all training instances $x^{(i)}$'s, the aim is to maximize the total return. As a widely preferred classification algorithm, multinomial logistic regression can be customized to address the introduced EDCS formulation and its objective. To this end, we propose a mixed-integer programming model (MIP) that maximizes the total return on a training set by building a classification task. This formulation defines a linear function mapping the feature space into score values for each class. An instance is thus classified as the class corresponding to the highest score value. The coefficients of this mapping function are optimized to be used on the unseen test data.

Mixed-integer programming models, on the other hand, are notoriously hard to solve and computationally complex such that no initial solution can be found for a very large data set within a reasonable time limit. In order to deal with this issue, we introduce Cost-Sensitive Logistic Regression (CSLR) that proposes a nonlinear approximation to our MIP model by optimizing its task using a gradient descent algorithm. Recently, blending machine learning and mixed-integer-programming together has received wide interest. There are several studies involving deep learning and gradient descent to solve optimization problems, as surveyed by Bengio *et al.* [11]; and also formulating a deep learning model as mixed-integer linear programming [12, 13]. CSLR bears the characteristics of both types of studies. Furthermore, we also empirically show that by providing a feasible solution taken from gradient descent based method as an initial solution, a mixed-integer programming solver can present significantly better optimization performance. Therefore, mixed-integer programming with the initial solution (MIP-WI) is also included in this work for computational experiments.

The rest of the thesis is organized as follows: Chapter 2 presents the background for the proposed EDCS formulation. In Chapter 3, the previous researches that introduce cost information during training are shown along with the works that study the intersection of machine learning and optimization. The proposed EDCS models and their formulations are presented in Chapter 4. Chapter 5 demonstrates the performance of the introduced approaches on the synthetic data sets in simulated experiments and on the real-world financial credit-scoring problem using realized financial returns. Finally, the conclusion of the thesis is drawn in Chapter 6.

2. BACKGROUND

As a core method in machine learning, regression makes use of linear predictor with a set of parameters θ to predict the value of $y^{(i)}$ for the i^{th} example $x^{(i)}$. Whereas linear regression uses a linear function $y = h_\theta(x) = \theta^\top x$ as hypothesis class, logistic regression feeds this linear predictor into a nonlinear function called *logistic (sigmoid)* to generate the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class. Hence, it builds a binary classification model on the training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ of N labeled examples, where the input features are $x^{(i)} \in \mathbb{R}^K$ and the labels take binary values, $y^{(i)} \in \{0, 1\}$. It's hypothesis takes the form:

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^\top x)}. \tag{2.1}$$

The probability values are then determined as $P(y = 1|x) = h_\theta(x)$ and $P(y = 0|x) = 1 - h_\theta(x)$. To build the best possible model, the performance of the hypothesis should be measured and optimized for parameters to get $\theta^\star$. For this purpose, logistic regression cost function can be written as:

$$J(\theta) = -\left[\sum_{i=1}^{N}\sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\} \log P(y^{(i)} = k|x^{(i)}; \theta)\right]. \tag{2.2}$$

This formulation assumes constant costs for all instances, hence, it is not suitable to be applied to EDCS problems. Correa Bahnsen *et al.* [14] managed to incorporate instance-based costs in logistic regression setting, where they introduced a cost matrix as in Table 2.1 that specifies varying costs corresponding to true positive, true negative, false positive and false negative for each instance.

Based on this cost matrix and a classifier f that generates predicted labels $d^{(i)} \in \{0, 1\}$, the objective function for binary EDCS learning can be created to minimize the

Table 2.1. Classification cost matrix for instance i.

	Actual Positive $y^{(i)} = 1$	**Actual Negative** $y^{(i)} = 0$
Predicted Positive $d^{(i)} = 1$	$C_{TP^{(i)}}$	$C_{FP^{(i)}}$
Predicted Negative $d^{(i)} = 0$	$C_{FN^{(i)}}$	$C_{TN^{(i)}}$

total costs instead of the negative loglikelihood of the loss function:

$$\widetilde{J}(\theta) = \sum_{i=1}^{N} \Big(y^{(i)} \big(d^{(i)} C_{TP^{(i)}} + (1 - d^{(i)}) C_{FN^{(i)}} \big) + (1 - y^{(i)}) \big(d^{(i)} C_{FP^{(i)}} + (1 - d^{(i)}) C_{TN^{(i)}} \big) \Big). \tag{2.3}$$

After constructing a cost-sensitive binary cost matrix for each instance, one can build an EDCS classification model that optimizes θ to minimize the Equation 2.3. In addition, using logistic regression formulation as classifier and plugging $h_\theta(x^{(i)})$ in the place of $d^{(i)}$ results in the minimum expected total costs, as Correa Bahnsen *et al.* [14] suggested. On the other hand, this formulation is only applicable to binary classification problems and cannot handle cases where there are more than two discrete outcomes. Therefore, this EDCS algorithm needs to be extended in order to be applied to multiclass problems.

Multinomial logistic regression presents a generalization of logistic regression to the multiclass classification case with the training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ where $y^{(i)} \in \{1, \ldots, C\}$ and C is the number of classes. Similar to aforementioned logistic regression model, multinomial logistic regression aims to estimate the probability $P(y = j|x)$ for each class label $j = 1, \ldots, C$. Hence, given the input x, the

hypothesis $h_\theta(x)$ becomes the following form:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=C|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{C} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(C)\top} x) \end{bmatrix} \tag{2.4}$$

where the term $\frac{1}{\sum_{j=1}^{C} \exp(\theta^{(j)\top} x)}$ makes the probability distribution sum to one for each instance. $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(C)} \in \mathbb{R}^K$ denote the parameters of the model associated with each class and can be written as single parameter matrix θ so that:

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \ldots & \theta^{(C)} \\ | & | & | & | \end{bmatrix}. \tag{2.5}$$

The cost function to measure how well $h_\theta(x)$ is performing for multinomial logistic regression is displayed as:

$$J(\theta) = -\left[\sum_{i=1}^{N} \sum_{j=1}^{C} \mathbb{1}(y^{(i)} = j) \log \frac{\exp(\theta^{(j)\top} x^{(i)})}{\sum_{k=1}^{C} \exp(\theta^{(k)\top} x^{(i)})} \right]. \tag{2.6}$$

To measure how well $h_\theta(x)$ is performing, the objective function tackles the minimization of the negative multinomial loglikelihood of the loss function. Making multinomial logistic regression formulation cost-sensitive, on the other hand, presents some challenges. One option is creating a cost matrix similar to Table 2.1, but now in multiclass setting. However, the cost matrix should be known before learning takes place and it can be hard to determine the elements of it for each instance as the number of classes increases. Therefore, the cost structure should go through a modification that removes the need for relative costs between class pairs in order to reduct the cost matrix. To this end, it can be assumed that each class j is accompanied with return values r_j if classified and a single returns matrix can be created as following:

Table 2.2. Returns matrix for example-dependent cost-sensitive learning.

Example	**Class 1**	$\cdots$	**Class C**
1	$r_1^{(1)}$	$\cdots$	$r_C^{(1)}$
$\vdots$			
i	$r_1^{(i)}$	$\cdots$	$r_C^{(i)}$
$\vdots$			
N	$r_1^{(N)}$	$\cdots$	$r_C^{(N)}$

EDCS classification problem can be then reformulated such that $x^{(i)} \in \mathbb{R}^K$ is a feature vector and $r^{(i)} \in \mathbb{R}^C$ is a known vector of returns, together forming the training set $\{(x^{(1)}, r^{(1)}), \ldots, (x^{(N)}, r^{(N)})\}$. Using multinomial logistic regression, the objective function takes then the following form with estimated probabilities:

$$R(\theta) = \sum_{i \in \mathbb{S}} r^{(i)\top} h_\theta(x^{(i)}). \tag{2.7}$$

Maximizing the Equation 2.7 would yield the expected total return of the model with the optimized parameters $\theta^\star$. Besides, one would also be interested in maximizing total returns as in the objective function depicted in Equation 2.3 without incorporating probability values, which is rather easy to be modeled using a linear mapping function and may present a clearer projection of test returns.

Say $d^{(i)} \in \mathbb{R}^C$ denotes the decision variable vector where each element is binary and $d_j^{(i)} = 1$ if the instance i is classified as j and 0 otherwise. Defining a classifier $f_\theta(x) = d$, the objective function to maximize the total returns would then become as following over the set of instances $\mathbb{S}$:

$$\widetilde{R}(\theta) = \sum_{i \in \mathbb{S}} r^{(i)\top} d^{(i)}. \tag{2.8}$$

Armed with this formula for the objective function, one can maximize it by optimizing for θ and build an EDCS classification model.

3. RELATED WORK

In this chapter, we first show the previous studies on cost-sensitive learning that use cost information in the learning phase. Then, the contributions to the intersection of machine learning and optimization are introduced.

3.1. Cost-Sensitive Learning Approaches

In literature, there have been several algorithms incorporating cost information during the learning process, but some of them are only applicable on the binary classification case whereas some others take the class-dependent costs into account, not the example-dependent ones.

An example of researches focusing on class-dependent costs is presented in [15]. Introducing class-dependent cost vectors, the authors made use of multilayer feedforward neural networks and compared the performances of four different modifications on the backpropagation learning algorithm when using the cost information: modifying the probability estimations, adapting the output of the network, adapting the learning rate and modifying the loss function. In their multiclass cost-sensitive deep learning model, Chung *et al.* [16] incorporated cost information while using autoencoders in pretraining and calculating a novel loss function in training step so that the algorithm pays attention to the varying costs not only in pretraining but also during training of the model. Khan *et al.* [8] created a class-dependent cost matrix and stacked it between the output layer and loss function in deep learning architecture and also proposed an optimization step for this matrix. These studies are successful in terms of introducing cost sensitivity during training in multiclass classification setting; on the other hand, the costs are not instance-based. Similarly, Fan *et al.* [17] and Masnadi-Shirazi and Vasconcelos [18] created class-dependent cost-sensitive boosting tree algorithms.

In addition to the aforementioned logistic regression model [14], decision tree and its ensembles are among the most used techniques in the literature of cost-sensitive learning with example-dependent costs. These methods are based on reformulating the splitting and pruning rules by proposing a cost-sensitive criterion. Bahnsen, in his researches [4, 19, 20], elaborated on cost-sensitive example-dependent learning for real-world credit scoring, churn and marketing data sets using realized financial costs. By creating a new measure called *savings*, defined as the cost of the algorithm versus the cost if no algorithm is used, Bahnsen managed to include instance-based costs into the decision tree algorithm and built smaller trees in comparison to standard trees. The ensembles of these trees make the algorithm even perform better. Although these works are very informative about the difference between class and example-dependent costs and how the costs can be introduced during training, they are limited to binary classification problems and require creating a cost matrix for each instance.

Furthermore, Sahin *et al.* [21] introduced a cost-sensitive decision tree approach for fraud detection and develop a new metric called *Saved Loss Rate (SLR)*, the saved percentage of the potential financial loss. The varying costs among examples are regarded while selecting the splitting attribute at each nonterminal node. Similarly, Aodha and Brostow [22] designed a cost-sensitive decision tree model by proposing a novel impurity measure that reformulates the splitting criterion such that the difference of between examples' cost is taken into account instead of their constant total cost. Being applicable to multiclass cases, this approach is applied to computer vision tasks.

A remarkable drawback of most of the existing cost-sensitive models is that they disregard maximizing the total returns of the classification and use the mainstream performance measures *accuracy, recall, precision, F Score and the area under the curve (AUC)* or some other domain specific metrics proposed in the corresponding papers. These performance metrics, on the other hand, are based on the counts of *True Positives, True Negatives, False Positives and False Negatives*. As shown explicitly in the example of credit scoring for financial institutions, the aim of the classification is to maximize the total profits in EDCS problems, which cannot be measured by the afore-

mentioned performance metrics. Hence, a new metric is required to draw inferences from cost-sensitive learning algorithms as Correa Bahnsen [4] proposes *savings*. Our proposed approach not only fill this gap by building a model that maximizes the total returns, but also extends cost-sensitive learning to be example-dependent while being applicable on multiclass data sets.

3.2. Machine Learning to Solve Mixed Integer Programming

Recent studies combine deep neural networks and mixed-integer programming in several ways. Arora *et al.* [23] and Serra *et al.* [24] formulated deep neural networks (DNNs) with rectified linear unit (ReLU) activation function as a mixed-integer programming model and studied the bounding techniques for linear regions. Since training in linear programming requires immense computational power for large set of data and may lead to overfitting for small sets, some researches pretrained the weights and focused on verification tasks on mixed-integer programming and generated adversarial examples in order to gauge the network's vulnerability to them. Fischetti and Jo [12] addressed a 0-1 mixed-integer programming model for modelling multilayer perceptron with ReLUs, and used pretrained weights to create adversarial data to fool the network. In a similar way, Tjeng *et al.* [25] improved this model in terms of solving time and also introduced more complex networks with convolutional layers in MIP setting to discuss the robustness of a neural network (NN) model. Lastly, Anderson *et al.* [13] developed strong mixed-integer programming formulations for trained neural networks with rectified linear units, showing the strength of their model with proofs and the success in solving time. They evaluated how robust the pretrained model is against the perturbation in input data. Besides, the studies that train the whole model using mixed-integer linear programming include the work of [26]. They modeled and trained Binarized Neural Network (BNN), where the weights and activations are restricted to the set $\{-1, +1\}$, as mixed-integer programming and constraint programming, and stated that if training data is limited, their approach generalize better than gradient descent based optimization. Similarly, Thorbjarnarson and Yorke-Smith [27] worked on BNNs with sign activation function and improved the previous model to be capable

of training more data in MIP setting and built an objective function of maximizing the accuracy.

On the other side, there are also significant works that use DNNs to solve known mixed-integer programming problems efficiently. Bengio *et al.* [11] surveyed the previous contributions and stated the methodology of how machine learning (ML) can be made use of to solve combinatorial optimization (CO) problems, such as the Traveling Salesman Problem. They emphasized two motivations for using ML to solve CO problems, finding approximations and discovery of new policies, and also exhibited three templates that how ML get involved in optimization tasks: In *end to end learning*, an ML model is trained to directly find feasible/optimal solutions for a given problem. While Vinyals *et al.* [28] and Bahdanau *et al.* [29] brought forward encoder-decoder format and attention mechanism to deal with TSPs, Bello *et al.* [30] suggested reinforcement learning and Kool *et al.* [31] made use of graph neural networks for solving these problems. Furthermore, ML can be used to deliver more insights of the problem to a CO algorithm in *learning to configure algorithms*. Kruber *et al.* [32] focused on whether learning helps determine the need for decomposition to solve the problem faster. Similarly, Bonami *et al.* [33] benefited from machine learning to find if linearizing speeds up the CO algorithm. Lastly, Bengio *et al.* [11] introduced the third template: *Machine learning alongside optimization algorithms*, that the CO algorithm calls an ML model once and again to feed the algorithm with lower end decisions such as branching decisions at the nodes in branch-and-bound algorithm [34, 35]. In addition, whether to call heuristics to find feasible solutions in B&B nodes can also be learned [36].

Based on the presented templates, the proposed CSLR approach belongs to *end to end learning*, where an ML model is trained to find a solution to an existing mixed-integer programming formulation. In addition, MIP-WI model can be considered to follow *learning to configure algorithms* principle, demonstrating how providing an initial solution may reduce the total number of operations, hence expediting the optimization process.

4. EXAMPLE-DEPENDENT COST-SENSITIVE LEARNING MODEL

This chapter introduces our EDCS learning framework. The proposed method builds a general mathematical formulation as a mixed-integer programming model that maximizes the total returns in a given EDCS problem. After introducing the approach, the thesis will focus on how to make use of gradient descent as an approximation to the formulated MIP model.

4.1. Mixed-Integer Programming For Cost Sensitive Learning

Recall the reformulated EDCS classification problem where $x^{(i)} \in \mathbb{R}^K$ is a feature vector, $r^{(i)} \in \mathbb{R}^C$ is a vector of returns and $d^{(i)} \in \mathbb{R}^C$ denotes the decision variable taking binary values. Assuming all return vectors $r^{(i)}$'s are fully known, the proposed approach is based on assigning a score value $p_j^{(i)}$ for each labeling decision $d_j^{(i)}$ and finding the maximum score value $\max_{j \in \mathbb{J}} p^{(i)}$. Then, the instance is assigned the class with the maximum score. To determine these values, a score function f is defined as:

$$f \colon x^{(i)} \rightarrow p_j^{(i)}, \quad i \in \mathbb{S}, \quad j \in \mathbb{J}. \tag{4.1}$$

The scoring scheme is outlined in Equation 4.2 if a linear mapping function is used similar to the aforementioned logistic regression model, but without transforming the scores into probabilities:

$$p = \theta^\top x. \tag{4.2}$$

The class assignment for decision variables d is done based on the maximum score value as shown in:

$$d_j^{(i)} = \begin{cases} 1, & \text{if } p_j^{(i)} = \max_{j \in \mathbb{J}} p^{(i)} \\ 0, & \text{otherwise} \end{cases} \qquad \forall i, j, \tag{4.3}$$

and

$$p_j^{(i)} \neq p_t^{(i)}, \quad \forall i, \quad j \in \mathbb{J}, \quad t \in \mathbb{J} \setminus \{j\}\,. \tag{4.4}$$

Writing the described scoring and classification scheme as linear programming requires several constraints to be introduced. Whereas it is straightforward to write a formulation that finds the maximum of a given set and assign its decision variable to "1", a more difficult challenge is formulating constraints that satisfy the Equation 4.4, which ensures that the scores corresponding for an instance take distinct values. They must be different from one another by at least a preconcerted margin so that a possible tie is prevented when determining the maximum one. Thus, we achieve the following model:

- Scalars
  - (i) lb, ub : lower bound and upper bound, indicate the range between which the scores take values.
  - (ii) ϵ : margin, a small number that determines the minimum margin between the score values corresponding to an instance.

- Indices and Sets
  - (i) Instances: $i \quad i \in \mathbb{S} = \{1, \ldots, N\}$
  - (ii) Classes: $j \quad j \in \mathbb{J} = \{1, \ldots, C\}$
  - (iii) Classes compared with class j: $t \quad t \in \mathbb{T}_j = \{(j+1), \ldots, C\}$
  - (iv) Features: $k \quad k \in \mathbb{K} = \{1, \ldots, K\}$

- Decision Variables
  - (i) $d_j^{(i)}$: Binary, 1 if the instance i is predicted as class j, 0 otherwise
  - (ii) $q_{jt}^{(i)}$: Binary, 1 if the score of class j is greater than the score of class t, 0 otherwise
  - (iii) $p_j^{(i)} : lb \leq p_j^{(i)} \leq ub, p_j^{(i)} \in \mathbb{R}$, the score of the class j for the instance i
  - (iv) $\theta_k^{(j)} : -\infty < \theta_k^{(j)} < \infty, \theta_k^{(j)} \in \mathbb{R}$, the coefficient of the class j for the feature k

- Parameters
  - (i) r : the returns matrix, $r \in \mathbb{R}^{C \times N}, r_j^{(i)}$ is the return of classifying the instance i as j
  - (ii) x : the data set x, $x \in \mathbb{R}^{K \times N}$

- The Model

$$\max \quad \widetilde{R}(\theta) = \sum_{i \in S} (r^{(i)\top} d^{(i)}) \tag{4.5a}$$

$$\text{s.t.} \qquad p_j^{(i)} - \theta^{(j)\top} x^{(i)} = 0 \qquad \forall i,\ \forall j, \tag{4.5b}$$

$$p_j^{(i)} - p_t^{(i)} - (ub - lb) q_{jt}^{(i)} \leq -\epsilon \qquad \forall i,\ j \in \mathbb{J} \setminus \{C\},\ t \in \mathbb{T}_j, \tag{4.5c}$$

$$p_j^{(i)} - p_t^{(i)} + (ub - lb)(1 - q_{jt}^{(i)}) \geq \epsilon \qquad \forall i,\ j \in \mathbb{J} \setminus \{C\},\ t \in \mathbb{T}_j, \tag{4.5d}$$

$$\sum_{t=j+1}^{C} q_{jt}^{(i)} - \sum_{t=1}^{j-1} q_{tj}^{(i)} - d_j^{(i)} \leq (C - j - 1) \qquad \forall i,\ \forall j, \tag{4.5e}$$

$$\sum_{j \in J} d_j^{(i)} = 1 \qquad \forall i, \tag{4.5f}$$

$$-\infty < \theta_k^{(j)} < \infty \qquad \forall j,\ \forall k, \tag{4.5g}$$

$$\theta_k^{(j)} \in \mathbb{R} \qquad \forall j,\ \forall k, \tag{4.5h}$$

$$-lb \leq p_j^{(i)} \leq ub \qquad \forall i,\ \forall j, \tag{4.5i}$$

$$p_j^{(i)} \in \mathbb{R} \qquad \forall i,\ \forall j, \tag{4.5j}$$

$$d_j^{(i)} \in \{0, 1\} \qquad \forall i,\ \forall j, \tag{4.5k}$$

$$q_{jt}^{(i)} \in \{0, 1\} \qquad \forall i,\ \forall j,\ t \in \mathbb{T}_j. \tag{4.5l}$$

(i) Equation 4.5a refers to the *objective* function in this mixed-integer programming model, maximizing the total returns.

(ii) In Constraint 4.5b, the linear scoring function f is displayed where the score value $p_j^{(i)}$ of the instance i for the class j is determined.

(iii) In Equations 4.5c and 4.5d, pairwise comparisons between the scores are made in the following way: Given the sets $j \in \mathbb{J} \setminus \{C\}$ and $t \in \mathbb{T}_j = \{(j+1), \ldots, C\}$, the score $p_j^{(i)}$ is compared with $p_t^{(i)}$ for each instance. While Constraint 4.5c makes the decision variable $q_{jt}^{(i)} = 1$ if $p_j^{(i)} > p_t^{(i)}$, Constraint 4.5d prevents $q_{jt}^{(i)} = 1$ if $p_j^{(i)} < p_t^{(i)}$ and forces $q_{jt}^{(i)} = 0$. In addition, ϵ in the RHS ensures the distinctness of the scores.

(iv) According to pairwise comparisons made in the previous constraints, Constraint 4.5e forces the labelling decision variable $d_j^{(i)} = 1$, demonstrating $p_j^{(i)}$ is the maximum among $p^{(i)}$.

(v) Equation 4.5f ensures that each instance i is labeled as only one class, j.

(vi) The remaining constraints are the cardinality constraints.

Solving this MIP model ends up with the optimized coefficients $\theta^\star$, where $\theta_k^{(j)}$ represents the weight on feature k for the decision j. Hence, one can make predictions for the unseen test data using these coefficients and finding the greatest score for each instance. On the other hand, this formulation brings about a remarkable drawback: As the number of instances or classes gets larger, the problem would become very complex and impossible to solve because of the compelling nature of mixed-integer programming. Therefore, an easy and fast way to tackle the problem should be also introduced to find incumbent solutions.

4.2. Cost-Sensitive Logistic Regression

Neural networks can be customized to represent the proposed MIP model and are fast in finding feasible solutions using gradient descent based optimization. Despite requiring hyperparameter tuning, gradient descent and its variants are fairly easy and fast methods to tackle optimization problems.

Recall *end to end learning*, where an ML model is trained to find a solution to an existing linear programming formulation. Accordingly, our aim is to reformulate EDCS mixed-integer linear programming model to make use of deep learning frameworks. To this end, here we introduce *Cost-sensitive Logistic Regression* (CSLR), a nonlinear approximation of the MIP model for EDCS learning.

The idea of reformulating the MIP model as a neural network stems from the scoring mechanism described in Equation 4.2, which is the same as the layer-wise operation in a neural network without an activation function, where the output is the weighted sum of the inputs: $p_j^{(i)} = \theta^{(j)\top} x^{(i)}, i \in \mathbb{S}, j \in \mathbb{J}$. Without adding bias term, the scores can be thus regenerated with a single layer perceptron with C nodes in the output layer, where the matrix θ, $\theta \in \mathbb{R}^{C \times K}$, represents the weights.

As in the MIP model, the next step is comparing the scores and determining the class with the maximum score for labelling decisions. The approximation of the class assignment variable d in MIP is achieved by a modification on the output layer. For instance, the outputs can be converted into probabilities using *softmax* activation function. Multiplication of the probabilities with the returns leads to the objective of maximizing the expected returns. Instead, the logits p_j of softmax, the vector of raw predictions the last layer generates, is multiplied with a large number M, so that the outputs of this function will be pushed towards 1 and 0, where the largest score value approximates to 1, and the others to 0 regardless of the number of classes. At the end of the layer, the predictions d_j take values as following:

$$d_j = \frac{\exp(Mp_j)}{\sum_{t=1}^{C} \exp(Mp_t)}, \quad j \in \mathbb{J}. \tag{4.6}$$

Figure 4.1 shows how the outputs of the softmax function changes as M increases for the binary classification case and Figure 4.2 depicts the visualization of CSLR as a single layer perceptron, where the activation function is denoted by *softmax* and applied on scores p of the unit to form its output $d = softmax(Mp)$. $x_1, \ldots, x_K$ represent input from one instance. All inputs are mapped onto the score values $p = \theta^\top x$

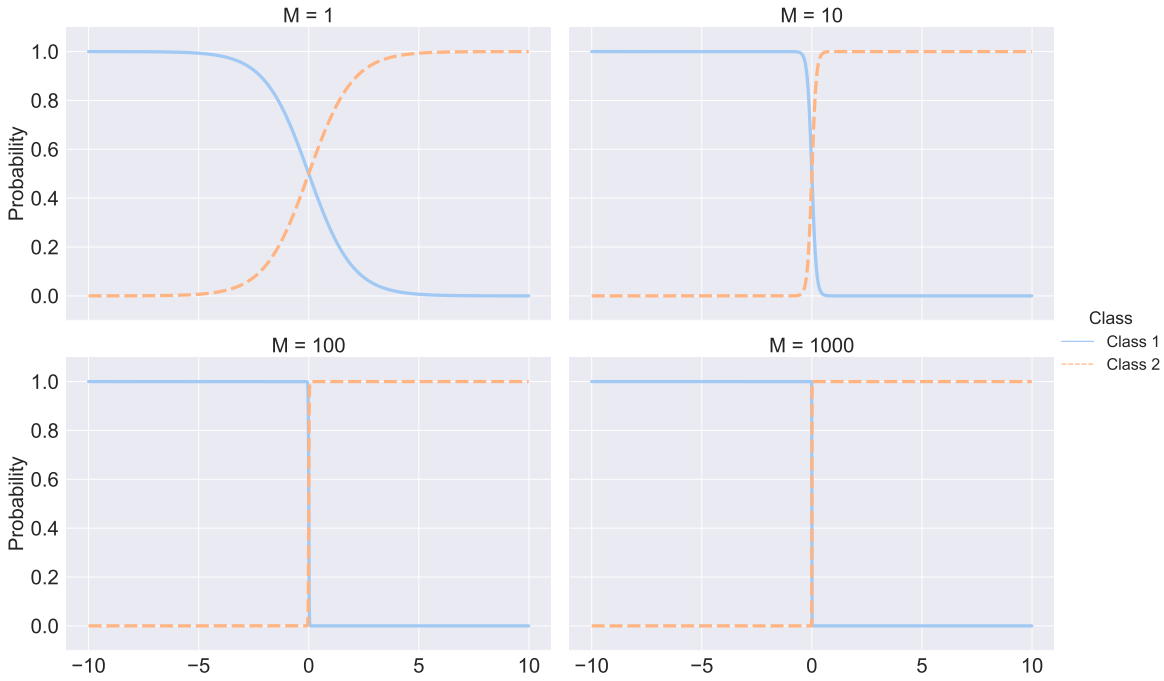

Figure 4.1. Softmax function with varying M values.

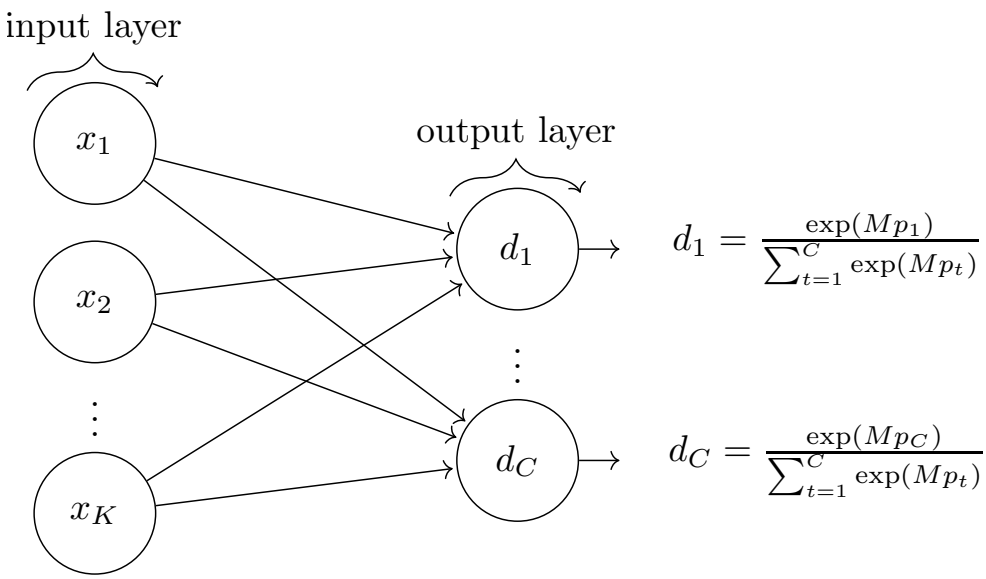

Figure 4.2. Single layer perceptron and its components.

using the propagation rule. The loss function is customized such that the returns $r^{(i)}$'s are multiplied with the predictions $d^{(i)}$'s. The loss value becomes then the negative of total return over the training set S, $loss = -\sum_{i \in \mathbb{S}} \mathbf{r}^{(i)\top} d^{(i)}$. This way one can achieve the same objective of maximizing the total returns and the same scoring mechanism in a perceptron setting as in the mixed-integer programming model. Gradient descent can be used to minimize the loss function.

Unlike the proposed linear programming model, this approach is applicable from small to large scale of problems. Furthermore, having an incumbent solution very fast would bring about the idea of accelerating the main MIP formulation. As mentioned, a linear programming solver can be very slow to find an initial feasible solution giving a large-scale problem. Following the principle of *learning to configure algorithms* [11], providing the solutions from CSLR to the MIP model as an initial solution can help in terms of increasing the speed to maximize the total return in the MIP approach. Providing a tighter lower bound for the objective can make it possible for tree-based branch-and-bound method to conduct better pruning within a given time limit, hence increasing the efficiency in MIP. To this end, we propose an additional approach called mixed-integer programming with the initial solution (MIP-WI), which takes the updated parameters θ from the perceptron and feeds them into the MIP algorithm as an initial solution.

The main advantage of formulating the proposed optimization model in neural network architecture is its speed and control over the model. In addition, one can add more layers to increase model complexity and tackle complex EDCS learning problems and obtain better results. For the sake of simplicity and the full representation of the proposed MIP approach, single layer perceptron is preferred in this work.

5. COMPUTATIONAL EXPERIMENTS

In this chapter, EDCS problems from both synthetic and real-life cases are used to compare our proposals. The motivation behind these experiments is to build up an understanding of the previously underlined issues regarding EDCS learning. We test our claim that CSLR provides a fast and accurate way to deal with EDCS problems and whether the performance of the MIP model can be enhanced if an initial solution is provided. We also analyze how well the proposed approaches perform on tasks with varying costs among examples in both binary and multiclass classification settings when compared to cost-insensitive machine learning methods. Lastly, we compare our approaches with other known EDCS models presented in Chapter 3.1 for binary classification problems.

First, the optimization performances of our proposals CSLR, MIP-WI and MIP are compared. To formulate EDCS model as a single layer perceptron, *PyTorch* environment is preferred where each model is run for 1,000 epochs using Adam optimizer [37] and makes use of batch gradient descent. Then, the last updated weight values of the gradient descent based optimization is fed into the mixed-integer programming formulation as coefficients, which is solved in linear programming solver *Gurobi v9.0.1* [38]. If the initial solution creates infeasibility that the scores are out of the bounds or the difference between scores is less than the margin, instances that create this infeasibility are removed and the rest of the data is used. The same set of training data is also used in MIP to provide a robust comparison between the two mixed-integer programming based methods and check whether MIP-WI provides enhanced results, regarding both optimization and test performances.

The test performances of the proposed approaches are first analyzed when compared with some state-of-the-art machine learning models: Logistic Regression, Decision Trees and boosting trees, namely Xgboost [39]. These models are trained with default parameters in their respective libraries *scikit-learn* [40] and *xgboost* [39], except

that the maximum depth of tree-based algorithms is fixed at 8 to prevent building very deep trees. Finally, we compare our proposals with Logistic Regression approach in [14], Decision Tree model in [19] and the ensemble methods Random Forest (RF) and Random Patches (RP) in [20]. These models introduce example-dependent costs during learning phase and can be implemented using the open source cost-sensitive classification library *CostCla* [4, 41]. Because they require a cost matrix for each instance as depicted in Table 2.1, we prepare Table 5.1 for demonstration purposes. The costs of *True Positives* and *True Negatives* are 0 whereas the costs of *False Positives* and *False Negatives* are the negative of the returns $r_1^{(i)}$ and $r_0^{(i)}$, respectively. Note that these algorithms are only applicable on binary classification problems.

Table 5.1. EDCS cost matrix for instance i in the experiments.

	Actual Positive $y^{(i)} = 1$	**Actual Negative** $y^{(i)} = 0$
Predicted Positive $d^{(i)} = 1$	0	$-r_1^{(i)}$
Predicted Negative $d^{(i)} = 0$	$-r_0^{(i)}$	0

With regards to scalar values in MIP based approaches, the lower bound lb and the upper bound ub on the scores are set to be -100 and 100, respectively; and the minimum margin ϵ between scores is 0.01. Before assigning the big-M value in CSLR, analyzing the adjacent region of ϵ would be beneficial to pick the right the big-M. Figure 5.1 bears the argument out that $M = 1,000$ is large enough for the revised softmax to force the decision variable of the greatest score value to 1 and the others to 0 if the absolute margin between score values is at least 0.01. The graphs on the left side display how the outputs of softmax function change with respect to the difference between two score values for binary classification problem. For multiclass case with $C = 3$ on the right, the x axis denotes the minimum of the absolute differences between scores. For both cases, if the minimum margin between scores is at least 0.01 and $M = 1,000$, the revised softmax function results in 1 for the class with the largest score and 0 for the

others. Therefore, these values are kept fixed for all experiments.

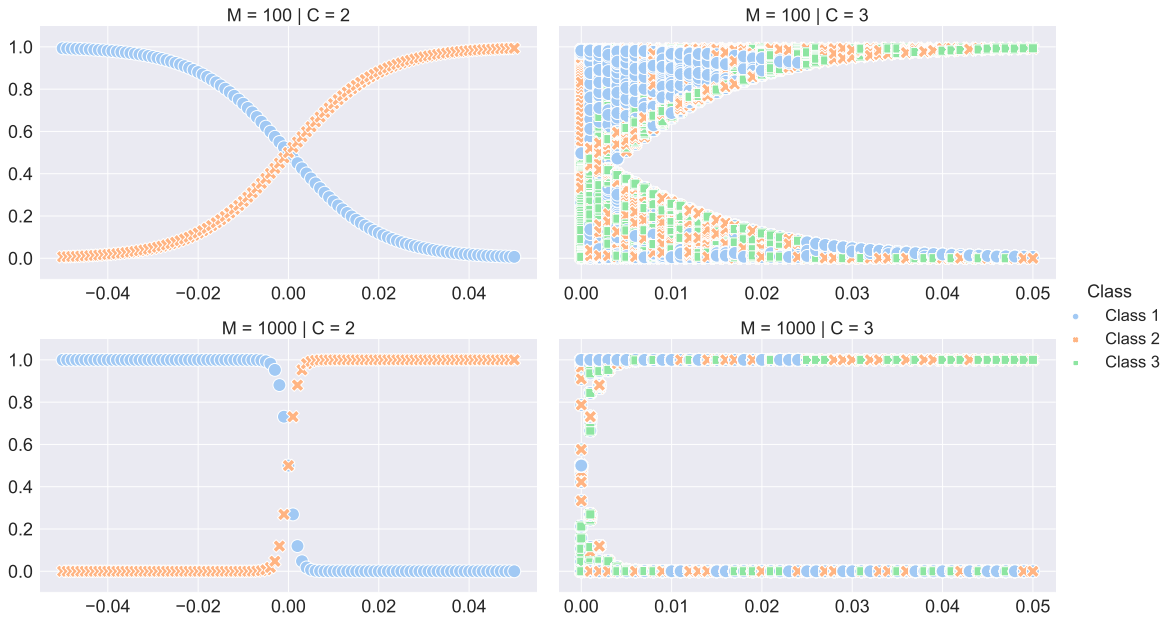

Figure 5.1. Softmax function with M = {100; 1,000} for both binary and multiclass cases.

For the linear programming methods, our strategy is built upon finding a feasible solution, instead of solving for optimal one, by optimizing the linear program to some extent within a reasonable time limit depending on the size of the problem. When comparing the optimization performances, the objective values of MIP and MIP-WI models are reported at the end of their time limits as well as their first feasible solutions and the time passed until the linear programming algorithm finds them. On the other side, the test performances of all models are compared according to *normalized total returns*, together with non-cost-sensitive metrics *accuracy* and *weighted F-Score*. Because it is complicated to interpret the total returns across different algorithms and data sets, they are normalized as proposed in [4,42], by dividing them by the theoretical maximum profit that is the total return if each instance is classified as the class with the highest return. Given the output vector $d^{(i)} \in \mathbb{R}^C$ of the classifier and the return

vector $r^{(i)} \in \mathbb{R}^C$, the *normalized return* for the training set $\mathbb{S}$ is shown as following:

$$\textit{Normalized Return}= \frac{\sum_{i\in\mathbb{S}} r^{(i)\top} d^{(i)}}{\sum_{i\in\mathbb{S}} \max_{j\in\mathbb{J}} r^{(i)}}. \tag{5.1}$$

The experiments are held using Python3.6.5 and the numerical results have been obtained on MSI MS-7A93 with an i9-7900X CPU and 32GB RAM and can be reproducible. The codes and the data sets can be found on the github page of the author [43].

5.1. Synthetic Data Sets

Simulation provides clear insights for real-world problems efficiently and effectively, especially when it is hard to collect data. In the context of EDCS learning, publicly available data sets are limited. For this purpose, synthetic data sets are generated for EDCS problems both from binary and multiclass setting.

The method to simulate the data sets is inspired by the works [44] and [45]. In our experiments, each data point $x_k^{(i)}$, $i = \{1, \ldots, N\}$ and $x^{(i)} \in \mathbb{R}^K$, is generated i.i.d. where odd-numbered feature k is distributed $\sim \mathbb{N}(0,\, 1)$ and even-numbered feature k is binary, sampled from Bernoulli with probability 0.5. Whereas generating the value of the returns is trivial, a mechanism that assigns them to different classes must be built. To this end, we define a function $f(\centerdot)$ consisting of two different parts, namely baseline and effect. Baseline defines the base outcome of the features and the effect function provides the differentiating part to generate the outcomes $o^{(i)} \in \mathbb{R}^C$ [45]. The highest return is assigned then to the class with the greatest outcome and the others have the same low return.

$$o_j^{(i)} = baseline(x^{(i)}) + effect_j(x^{(i)}), \quad i \in \mathbb{S}, \quad j \in \mathbb{J}. \tag{5.2}$$

In this thesis, two types of problems are considered when simulating the data sets: binary and 3-class classification.

(i) In the first setting, a binary cost sensitive classification problem is presented where number of feature K=25.

$$o_0(x) = x_1 + x_2 + x_7 + x_{10} + x_{12} + x_{17} + x_{24} \tag{5.3}$$

$$+ 5\mathbb{1}(x_{11} > -0.6) + |x_{19} \times x_{21}| + x_{24} \tag{5.4}$$

$$o_1(x) = x_1 + x_2 + x_7 + x_{10} + x_{12} + x_{17} + x_{24} + |x_{13} \times x_{15}| + x_{20} \tag{5.5}$$

(ii) The second setup presents a multiclass problem with 3 classes and 25 features.

$$o_0(x) = x_1 + x_2 + x_7 + x_{10} + x_{12} + x_{17} + x_{24} + 2 \tag{5.6}$$

$$o_1(x) = x_1 + x_2 + x_7 + x_{10} + x_{12} + x_{17} + x_{24} \tag{5.7}$$

$$+ 7\mathbb{1}(x_{23} > 0) + |x_3 \times x_{15}| + x_{22} \tag{5.8}$$

$$o_2(x) = x_1 + x_2 + x_7 + x_{10} + x_{12} + x_{17} + x_{24} \tag{5.9}$$

$$+ 7\mathbb{1}(x_{11} > -0.2) + |x_{13} \times x_{21}| + x_{20} \tag{5.10}$$

For both data set types, a linear baseline function and a piecewise linear effect function with a combination of identity, absolute value and constant parts are formulated on 25 features. Note that the effect function is also designed to create class imbalance within the data set. The outcomes are then standardized in both training and test sets as following:

$$o_j^{train} = \frac{o_j^{train} - mean(o_j^{train})}{std(o_j^{train})}, \quad o_j^{test} = \frac{o_j^{test} - mean(o_j^{train})}{std(o_j^{train})}, \quad j \in \mathbb{J}. \tag{5.11}$$

The reason to use the same mean and standard deviation of the training set on the test set is that classification tasks require creating the same setting as in training set when model testing takes place [46]. After standardizing, we add noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ to the outcomes in the training set. Before generating the returns, the return value corresponding to the class with maximum outcome is set to be a positive real number whereas others are negative which in turn decreases the total return if classified.

Table 5.2. The distribution of returns for instance i - binary case.

	$\mathbf{r_1^{(i)}}$	$\mathbf{r_2^{(i)}}$
if $\max_{j\in\mathbb{J}} o^{(i)} = o_1^{(i)}$	$\mathcal{U}(100, 500)$	$-\mathcal{U}(50, 150)$
if $\max_{j\in\mathbb{J}} o^{(i)} = o_2^{(i)}$	$-\mathcal{U}(50, 150)$	$\mathcal{U}(500, 2000)$

Table 5.2 demonstrates the distribution of the returns for the binary classification case and Table 5.3 shows how returns are sampled in 3-class problem setting in the experiments. Each return value is sampled from Uniform Distribution.

Table 5.3. The distribution of returns for instance i - 3-class case.

	$\mathbf{r_1^{(i)}}$	$\mathbf{r_2^{(i)}}$	$\mathbf{r_3^{(i)}}$
if $\max_{j\in\mathbb{J}} o^{(i)} = o_1^{(i)}$	$\mathcal{U}(500, 2000)$	$-\mathcal{U}(50, 150)$	$-\mathcal{U}(50, 150)$
if $\max_{j\in\mathbb{J}} o^{(i)} = o_2^{(i)}$	$-\mathcal{U}(50, 150)$	$\mathcal{U}(300, 500)$	$-\mathcal{U}(50, 150)$
if $\max_{j\in\mathbb{J}} o^{(i)} = o_3^{(i)}$	$-\mathcal{U}(50, 150)$	$-\mathcal{U}(50, 150)$	$\mathcal{U}(100, 300)$

The simulated experiments are designed such that the optimization performance of our proposals are compared for binary classification problems where the number of instance $N = \{1,000; 2,000; 3,000\}$ and for 3-class problems where the number of instances $N = \{250; 500; 1,000\}$. The test performances of all methods are then analyzed where $N_{test} = 20,000$ and $N_{train} = 3,000$ for binary cases and $N_{train} = 1,000$ for multiclass cases, respectively. The outcomes in the training sets are assigned the noise $\sigma^2 = 1$ and the time limit of MIP and MIP-WI methods is set to 5 minutes for each training set. A fixed learning rate of 10^{-2} is used for CSLR. Each experiment is replicated 50 times.

5.1.1. Results

Training results of the EDCS models in synthetic data sets are displayed in Figure 5.2. For each experiment type, we depict the objective values of MIP-WI and MIP with the training results from CSLR after 1000 epochs. We also show that how these objective values change during optimization by reporting the first feasible solution and

the one after 5 minutes. It can be seen that the performance of the linear programming models is heavily affected by the size of the data set. In all experiments, MIP-WI outperforms MIP in terms of their objective values after 5 minutes, with an increasing margin as the number of classes and instances get larger. CSLR, on the other hand, contributes to the most of this margin by being not negatively affected by the increase in the size of the data and achieves better results than MIP except when $N = 1000$ for the binary case and $N = 250$ for the multiclass case. Hence, the claim that CSLR can deal with the formulated optimization problem faster than MIP in large data sets is validated in the simulated experiments.

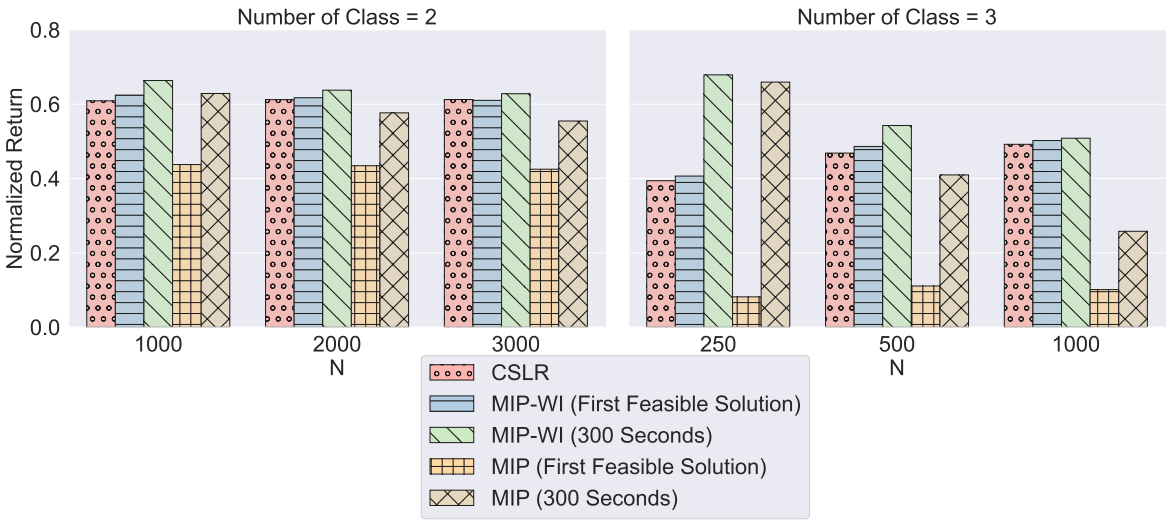

Figure 5.2. Optimization performance of EDCS approaches in synthetic data sets.

Table 5.4 shows the elapsed time until MIP-WI and MIP find their first feasible solutions along with the training times of CSLR. The first feasible solution of MIP-WI denotes the first improvement MIP-WI achieves on the initial results taken from CSLR. It is noticeable that 3-class data sets require more time than the binary ones although the former contains fewer instances. We also see that it takes more time to find feasible solutions as the number of instances gets larger. On average, MIP-WI achieves the first feasible solution later than MIP, which can be explained by the fact that CSLR has already found a good solution for MIP-WI and it is harder to improve the objective value the nearer the algorithm to the optimum.

Table 5.4. Computational times in seconds in synthetic data sets.

Number of Classes	**Number of Instances**	**Method**	**Time (s)**
$C = 2$	$N = 1000$	MIP-WI	3.3
		MIP	0.43
		CSLR	6.12
	$N = 2000$	MIP-WI	12.70
		MIP	1.89
		CSLR	10.58
	$N = 3000$	MIP-WI	32.76
		MIP	11.27
		CSLR	12.07
$C = 3$	$N = 250$	MIP-WI	7.50
		MIP	0.83
		CSLR	4.49
	$N = 500$	MIP-WI	23.26
		MIP	13.04
		CSLR	5.15
	$N = 1000$	MIP-WI	78.19
		MIP	30.49
		CSLR	8.81

Figure 5.3 illustrates the test results of EDCS models and other machine learning methods in binary data set where $N = 3,000$ and 3-class data set where $N = 1,000$. CSLR and MIP-WI present the best performance in the binary case and yields competitive results compared to Xgboost algorithm in 3-class classification. MIP, on the other hand, performs poorly compared with CSLR and MIP-WI, demonstrating testing performance can be negatively affected if the training data is not optimized sufficiently. Similarly, Decision Tree and Logistic Regression fall behind of CSLR, MIP-WI and Xgboost. For both cases, MIP-WI improves the test returns slightly upon CSLR; however, the difference is so small and can be negligible. Their training performances are

also very similar as can be seen in Figure 5.2. It can therefore be concluded that MIP-WI may not be worth implementing in large scale problems because of extra training costs. Furthermore, we also show that there is no correlation to be observed between normalized returns and other metrics accuracy and F score, e.g., Logistic Regression has the highest accuracy in binary classification, but its normalized return is fairly low compared to CSLR.

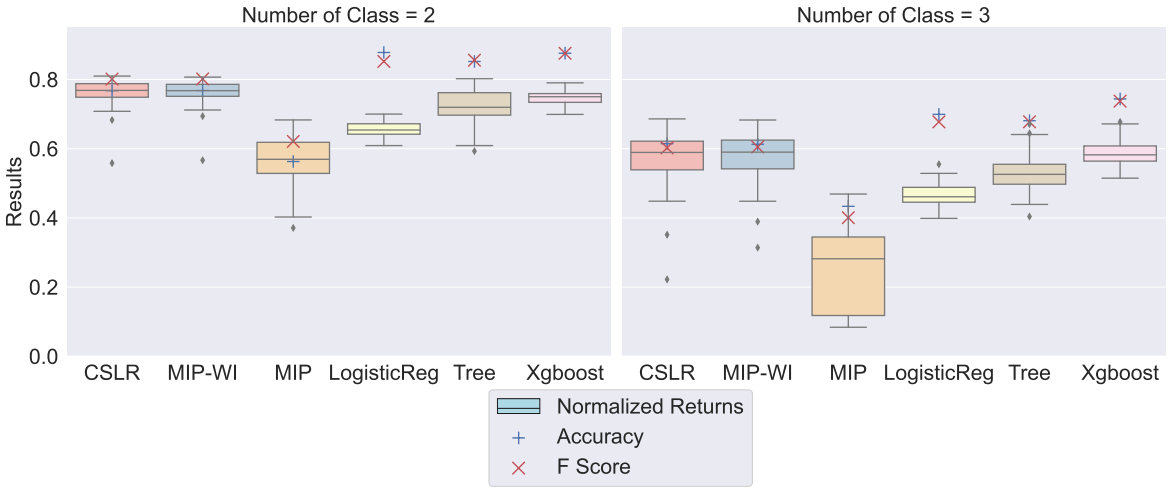

Figure 5.3. Test results in synthetic data sets.

5.1.2. Comparison with other EDCS Learning Models

In Figure 5.7, we compare the test results of our proposals with other known EDCS models on synthetic data sets. Since these models cannot handle multiclass data sets, only the binary problem with $3,000$ examples is considered for comparisons. Cost-sensitive Decision Tree approach in [19] stands out as the best approach among them in terms of normalized returns; on the other hand, our proposals CSLR and MIP-WI present better results by a significant margin. A remarkable point is that the ensemble models RF and RP in [20] perform poorly and result in approximately 56% of normalized returns on average, which is the same as classifying all examples as "0". In addition, our proposed Cost-sensitive Logistic Regression approach outperforms the Logistic Regression model in [14].

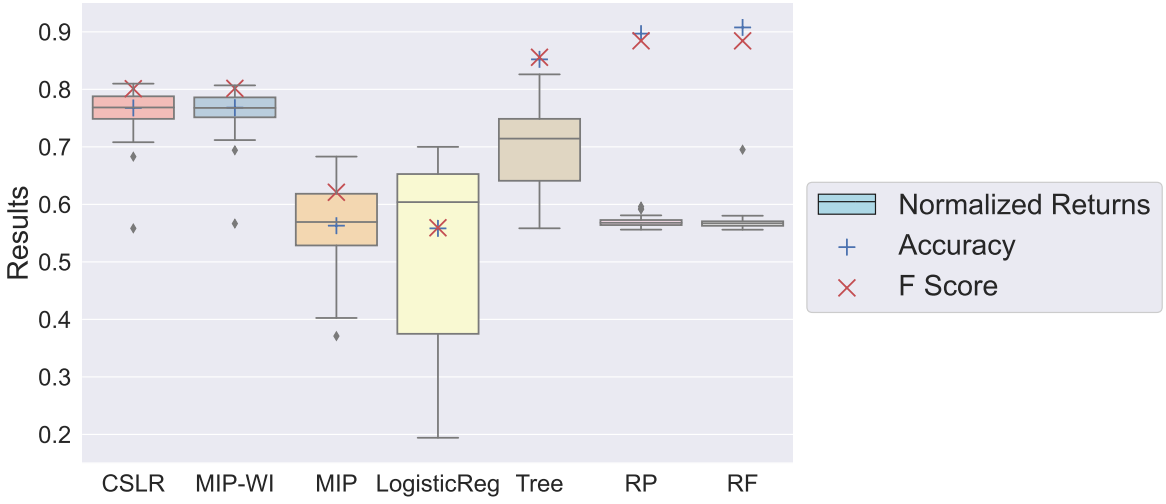

Figure 5.4. Comparison of our proposals with other known EDCS models - synthetic data sets.

5.2. Experiments on Real-Life Data - Credit Scoring

In this section, the proposed approaches are applied to real-life credit scoring problem. To this end, we obtain Bank Credit Data Set from a financial institution. The data set demonstrates the approved loan applications from the customers of a bank including how much the applicants paid in return and the bank's profit from these transactions. It also shows a customer's requested loan amount and any other useful personal information such as how many credit cards does a customer have and how much are their limits, whether the applicants defaulted before or have debts, etc. Thus, all the necessary information showing customer's financial situation is contained in order that a decision-making model for the bank can be built when granting a loan.

In this experiment, for the linear models, Logistic Regression and our proposed EDCS approaches, the data is centered and scaled between zero and one as the range of the values of features varies among each other drastically. When working with gradient descent, scaling makes optimization converge [47]. It also increases the interpretability of the coefficients and makes sensitivity analysis possible for the relative changes of the coefficients of covariates on the score values.

Table 5.5. Returns matrix for Bank Credit Data Set.

	Bank Profit-Approved Loan	Bank Profit-Rejected Loan
Customer i	$pr^{(i)}$	$-pr^{(i)}$

Denoting the profit the bank makes from a customer as $pr^{(i)}$, $i = \{1, \ldots, N\}$, Table 5.5 shows how much the bank would get as return from it's decisions of approving or rejecting the loans. The first column shows the profit the bank made in the case of a granted loan, which also happened in reality, and the second column displays the opportunity cost for the bank if the loans were not granted. We simply make the opportunity cost equal to minus the profit for demonstration purposes.

The data set consists of 2562 examples, where the number of defaulters is 258, making the proportion of positive examples approximately 10%. The data adds up a total of 61 features, numeric or categorical, which are one-hot encoded in preprocessing. To create a robust environment, the data is split into 10 training and validation folds and the models are trained and evaluated on the same sets. The experiment is replicated 5 times with re-sampled training and validation folds. The time limit of MIP and MIP-WI methods is set to 10 minutes for each training set and a fixed learning rate of 10^{-3} is used for CSLR.

5.2.1. Results

Figure 5.5 compares the optimization performance of MIP-WI and MIP in the Bank Credit Data Set. The average training time of CSLR is 5.58s. Unlike simulated experiments, MIP-WI manages to find the first feasible solution faster (3.92s) than MIP (6.24s) on average, indicating the long initialization time mixed-integer programming algorithm may have and how CSLR can surpass it by providing initial solutions. After 10 minutes, MIP-WI has a higher objective value on average than MIP, but the drastic difference between their first feasible solutions starts to decrease as the optimization process proceeds. In addition, the relative increase in objective values for both models

also starts shrinking.

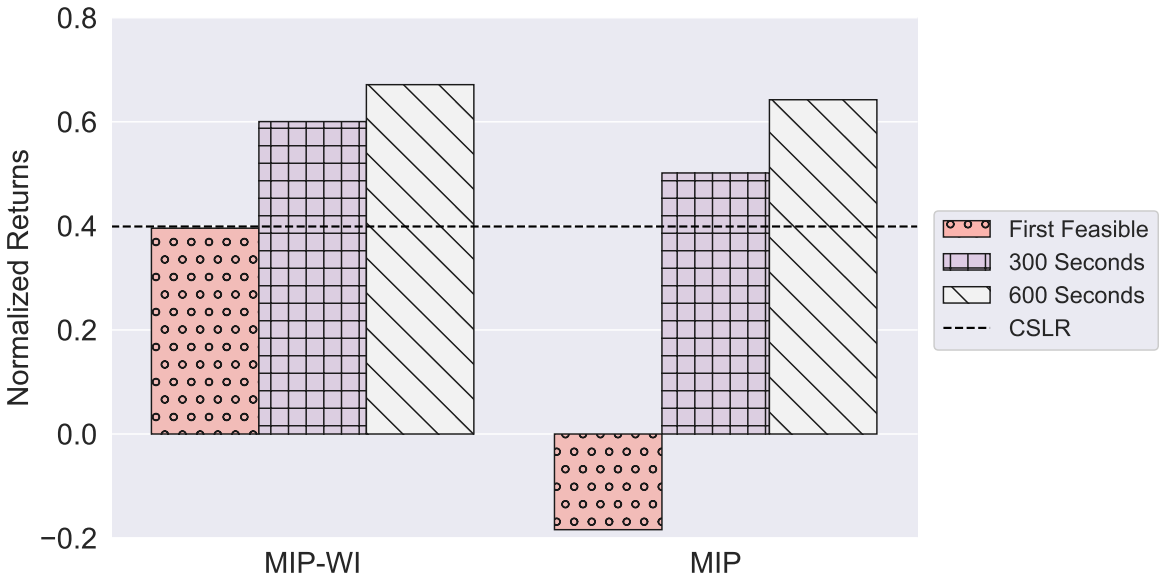

Figure 5.5. Optimization performance of EDCS approaches in Bank Credit Data Set.

In Figure 5.6, test results of the proposed EDCS learning approaches on the Bank Credit Data Set are displayed along with the ones from state-of-the-art machine learning methods. The dashed horizontal black line represents the realized normalized return the bank achieved. It is easily noticeable that on average all methods perform better than the bank. Similar to simulated experiments, normalized returns seem to have no correlation with accuracy and F Score. Another remarkable finding is that EDCS learning models outperform all state-of-the-art machine learning models, MIP-WI being the best one by a significant margin. Note also that in this experiment MIP presents better test performance than CSLR, at the expense of 10 minutes training time.

Overall, the results exhibit that credit scoring problem can be tackled by algorithms with EDCS learning nature, such as the formulation presented in this thesis.

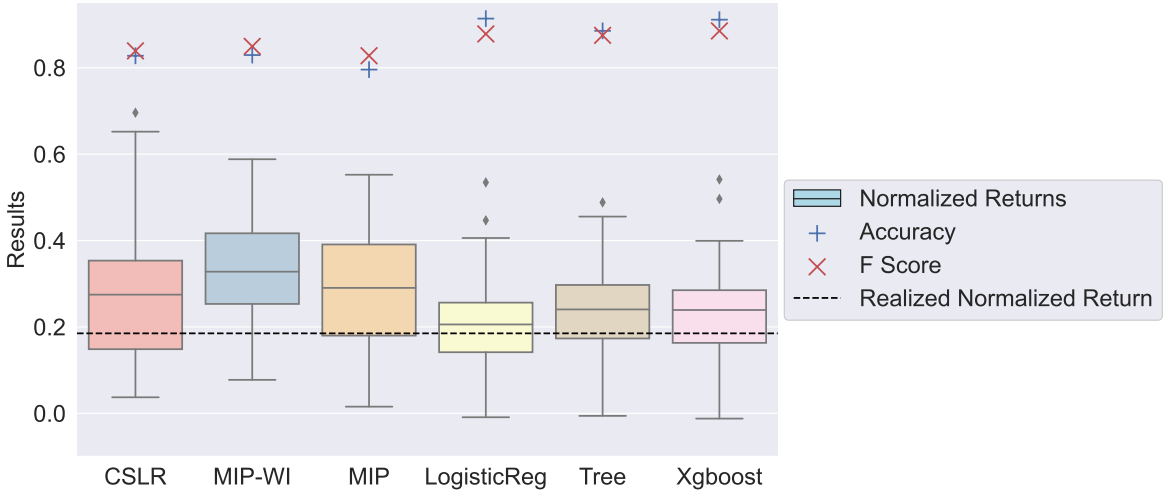

Figure 5.6. Test results in Bank Credit Data Set.

5.2.2. Comparison with other EDCS Learning Models

In Figure 5.7, we compare the test results of our proposals with other known EDCS models on Bank Credit Data Set. Unlike simulated experiments, tree based EDCS learning algorithms [19, 20] present better performance and outperform the realized normalized return the bank achieved. Once again, our proposed Cost-sensitive Logistic Regression overcomes the Logistic Regression model in [14], which shows the success of our approach to EDCS problems. EDCS Decision Tree model presents competitive performance when compared with CSLR, but MIP-WI still achieves the highest normalized returns by a significant margin.

5.3. Discussion

In our experiments, we demonstrate that MIP-WI achieves better optimization performance than MIP within the given time limits. The difference between their objective values is significant in the early phase of the optimization process. It also increases with the growing size of the training examples and the number of classes. Given a large scale problem, as in simulated experiments where $N = 3,000$ for the binary problem and $N = 1,000$ for the multiclass problem, the contribution of MIP-

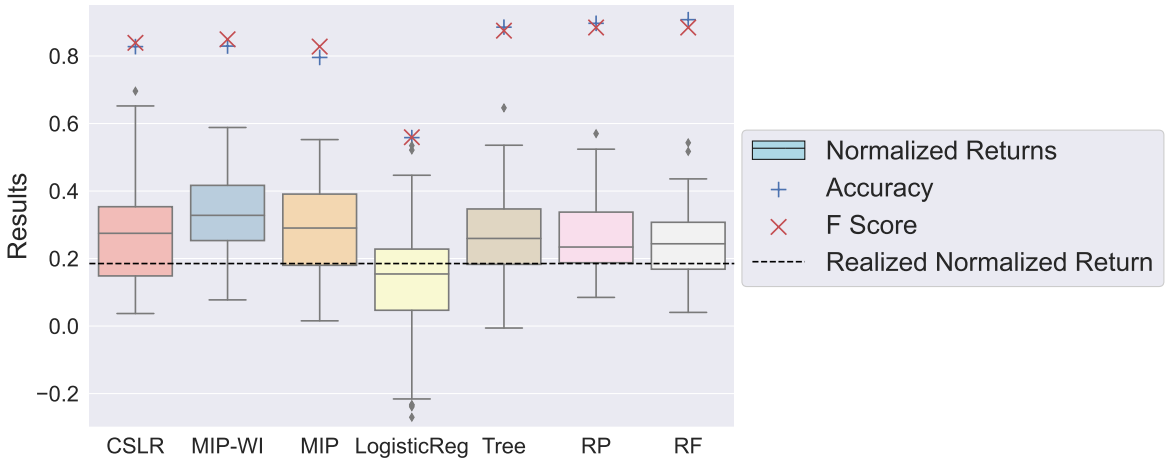

Figure 5.7. Comparison of our proposals with other known EDCS models - Bank Credit Data Set.

WI onto the result of CSLR may be negligible and MIP struggles to produce feasible solutions with objective value closer to one from CSLR. Hence, MIP and MIP-WI seem to be not a rational method for training if the size of the problem is large. The performance of CSLR, on the other hand, is not subject to a decrease for big data sets and even increases as the number of instances gets larger in multiclass cases.

Our EDCS models present outstanding test results, outperforming other ML methods in most of the experiments. In fact, the results can be assumed to favor the tree based models at first glance because the data sets both in simulated experiments and credit scoring problem are highly structured, and Xgboost stands out as one of the most powerful tool to deal with them. Thus, we show the importance of introducing instance based costs/returns in a machine learning algorithm for EDCS problems. In addition, our proposals show more robust and more successful results when compared to some of other EDCS learning models in literature.

Another important point is that neither CSLR nor other ML methods are gone through hyperparameter tuning process. One can increase both training and testing performance of CSLR by tuning hyperparameters such as learning rate and mini-batch

size, hence also obtaining a better result from MIP-WI. This process, on the other hand, can be time consuming and is left as a future work. As mentioned, CSLR can also capture more complex decision boundaries by having more than one layer or a different architecture. Despite requiring a relatively long time for optimization, MIP does not own any hyperparameter to tune. One can easily run the model in a linear programming solver and achieve feasible solutions for the coefficients on features if the size of the problem permits.

Finally, we provide evidence that the common performance metrics accuracy and F Score cannot be relied on to evaluate how much the total returns can be. This supports our argument that EDCS problems should not be evaluated by the metrics that are based on the counts of correct classifications and misclassifications.

6. CONCLUSION

In this thesis, we illustrate how EDCS learning problems can be successfully addressed if example-dependent costs/returns are incorporated in the learning algorithm. The aim of our approach is based on maximizing the total return of a given data set. For this purpose, we build a classification task that makes use of linear mapping function to assign score values for each class. An instance is thus classified as the class corresponding to the highest score value. This classification task is formulated as a mixed-integer programming model that may struggle to produce feasible solutions if the size of the data set is large. To address this, we propose a nonlinear approximation of this model, which benefits from gradient descent based optimization. We also assert that the performance of the mixed-integer programming model can be improved if initialized with solutions from the nonlinear model.

To test our claims and formulations, we obtained a credit scoring data set from a financial institution. Since publicly available data sets for EDCS learning is limited, we also generate synthetic data using simulation. Our approaches present significant results in synthetic and credit scoring data sets when compared to not only state-of-the-art machine learning algorithms, but also other known EDCS models. In addition, we also empirically show that the performance of the proposed mixed-integer programming model can be improved if initialized with the solutions from gradient descent based optimization. Being applicable on multiclass data sets, our proposals present a general strategy for EDCS problems and supply a significant contribution to the literature of EDCS learning.

The main contribution of this thesis is a unifying framework for utilizing machine learning in solving notoriously hard optimization problems, specifically in the context of instance-based learning. For this purpose, the proposed MIP model to solve EDCS formulation is improved by a nonlinear approximation. Future works may concern the adaptation of CSLR to more business-oriented problems and the performance im-

provement of CSLR itself. By using a linear mapping function to calculate individual score values, CSLR does not take into account of the nonlinear relationships between features. If nonlinearity is present in the data, some modifications on the model can be considered as long as the output layer is the same as proposed in this thesis. For instance, applying a kernel on the score function, adding more layers to the network, or adapting any other architecture fitting the data set may make the model achieve better results.

REFERENCES

1. Zadrozny, B. and C. Elkan, "Learning and Making Decisions When Costs and Probabilities Are Both Unknown", *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 204–213, 2001.

2. Elkan, C., "The Foundations of Cost-Sensitive Learning", *Seventeenth International Joint Conference on Artificial Intelligence*, pp. 973–978, 2001.

3. Wang, T., *Efficient Techniques for Cost-Sensitive Learning with Multiple Cost Considerations*, Ph.D. Thesis, University of Technology, Sydney, 2013.

4. Correa Bahnsen, A., *Example-Dependent Cost-Sensitive Classification with Applications in Financial Risk Modeling and Marketing Analytics*, Ph.D. Thesis, University of Luxembourg, Luxembourg, 2015.

5. Zadrozny, B., J. Langford and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting", *Third IEEE International Conference on Data Mining*, pp. 435–442, 2003.

6. Chawla, N. V., K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research*, Vol. 16, No. 1, pp. 321–357, 2002.

7. Zhou, Z.-H. and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 1, pp. 63–77, 2006.

8. Khan, S. H., M. Hayat, M. Bennamoun, F. A. Sohel and R. Togneri, "Cost-Sensitive Learning of Deep Feature Representations From Imbalanced Data", *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 8, pp. 3573–3587,

2018.

9. Zhou, Z.-H. and X.-Y. Liu, "On Multi-Class Cost-Sensitive Learning.", *Proceedings of the 21st National Conference on Artificial Intelligence*, pp. 232–257, 2006.

10. Correa Bahnsen, A., A. Stojanovic, D. Aouada and B. Ottersten, "Cost Sensitive Credit Card Fraud Detection Using Bayes Minimum Risk", *12th International Conference on Machine Learning and Applications*, pp. 333–338, IEEE, Miami, USA, 2013.

11. Bengio, Y., A. Lodi and A. Prouvost, "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon", *European Journal of Operational Research*, Vol. 290, No. 2, pp. 405–421, 2020.

12. Fischetti, M. and J. Jo, "Deep neural networks and mixed integer linear optimization", *Constraints*, Vol. 23, No. 11, pp. 296–309, 2018.

13. Anderson, R., J. Huchette, W. Ma, C. Tjandraatmadja and J. Vielma, "Strong mixed-integer programming formulations for trained neural networks", *Mathematical Programming*, Vol. 183, No. 6, pp. 3–39, 2020.

14. Bahnsen, A. C., D. Aouada and B. Ottersten, "Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring", *13th International Conference on Machine Learning and Applications*, pp. 263–269, IEEE, 2014.

15. Kukar, M. and I. Kononenko, "Cost-Sensitive Learning with Neural Networks", *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pp. 445–449, John Wiley & Sons, 1998.

16. Chung, Y., H. Lin and S. Yang, "Cost-aware Pre-training for Multiclass Cost-sensitive Deep Learning", *CoRR*, Vol. abs/1511.09337, 2015.

17. Fan, W., S. Stolfo, J. Zhang and P. Chan, "AdaCost: Misclassification Cost-

sensitive Boosting", *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, pp. 97–105, 1999.

18. Masnadi-Shirazi, H. and N. Vasconcelos, "Cost-Sensitive Boosting", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33, No. 2, pp. 294 – 309, 2011.

19. Correa Bahnsen, A., D. Aouada and B. Ottersten, "Example-Dependent Cost-Sensitive Decision Trees", *Expert Systems with Applications*, Vol. 42, No. 19, pp. 6609–6619, 2015.

20. Bahnsen, A. C., D. Aouada and B. E. Ottersten, "Ensemble of Example-Dependent Cost-Sensitive Decision Trees", *CoRR*, Vol. abs/1505.04637, 2015.

21. Sahin, Y., S. Bulkan and E. Duman, "A cost-sensitive decision tree approach for fraud detection", *Expert Systems with Applications*, Vol. 40, No. 15, pp. 5916–5923, 2013.

22. Aodha, O. M. and G. J. Brostow, "Revisiting Example Dependent Cost-Sensitive Learning with Decision Trees", *IEEE International Conference on Computer Vision*, pp. 193–200, 2013.

23. Arora, R., A. Basu, P. Mianjy and A. Mukherjee, "Understanding Deep Neural Networks with Rectified Linear Units", *arXiv e-prints*, p. arXiv:1611.01491, 2016.

24. Serra, T., C. Tjandraatmadja and S. Ramalingam, "Bounding and Counting Linear Regions of Deep Neural Networks", *Proceedings of the 35th International Conference on Machine Learning*, pp. 4558–4566, PMLR, 2018.

25. Tjeng, V., K. Xiao and R. Tedrake, "Evaluating Robustness of Neural Networks with Mixed Integer Programming", *arXiv preprint arXiv:1711.07356*, 2017.

26. Toro Icarte, R., L. Illanes, M. P. Castro, A. A. Cire, S. A. McIlraith and J. C. Beck,

"Training Binarized Neural Networks Using MIP and CP", *Principles and Practice of Constraint Programming*, pp. 401–417, Springer International Publishing, Cham, 2019.

27. Thorbjarnarson, T. and N. Yorke-Smith, "On Training Neural Networks with Mixed Integer Programming", *arXiv preprint arXiv:2009.03825*, 2020.

28. Vinyals, O., M. Fortunato and N. Jaitly, "Pointer Networks", C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, Vol. 28, Curran Associates, Inc., 2015.

29. Bahdanau, D., K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", *arXiv e-prints*, p. arXiv:1409.0473, 2014.

30. Bello, I., H. Pham, Q. V. Le, M. Norouzi and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning", *arXiv e-prints*, p. arXiv:1611.09940, 2016.

31. Kool, W., H. van Hoof and M. Welling, "Attention, Learn to Solve Routing Problems!", *International Conference on Learning Representations*, pp. 1–25, 2019.

32. Kruber, M., M. Lübbecke and A. Parmentier, "Learning When to Use a Decomposition", *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 202–210, 2017.

33. Bonami, P., A. Lodi and G. Zarpellon, "Learning a Classification of Mixed-Integer Quadratic Programming Problems", W.-J. van Hoeve (Editor), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 595–604, Springer International Publishing, Cham, 2018.

34. Lodi, A. and G. Zarpellon, "On learning and branching: a survey", *TOP*, Vol. 25, No. 2, pp. 207–236, 2017.

35. Andréung, S. Tanaka and K. Tierney, "Deep learning assisted heuristic tree search for the container pre-marshalling problem", *Computers and Operations Research*, Vol. 113, p. 104781, 2020.

36. Khalil, E., B. Dilkina, G. Nemhauser, S. Ahmed and Y. Shao, "Learning to Run Heuristics in Tree Search", *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, Vol. 12, pp. 659–666, 2017.

37. Kingma, D. P. and J. Ba, "Adam: A Method for Stochastic Optimization.", *CoRR*, Vol. abs/1412.6980, 2014.

38. Gurobi Optimization, L., "Gurobi Optimizer Reference Manual", `http://www.gurobi.com`, accessed in 02.01.2021.

39. Chen, T. and C. Guestrin, "XGBoost: A Scalable Tree Boosting System", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, ACM, New York, NY, USA, 2016.

40. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python", *CoRR*, Vol. abs/1201.0490, 2012.

41. Bahnsen, C., "Costcla: Python module for cost-sensitive machine learning", `https://pypi.org/project/costcla/`, accessed in 02.01.2021.

42. Whitrow, C., D. Hand, P. Juszczak, D. Weston and N. Adams, "Transaction aggregation as a strategy for credit card fraud detection", *Data Mining and Knowledge Discovery*, Vol. 18, No. 1, pp. 30–55, 2009.

43. Temizoz, T., "Cost-Sensitive-Learning", `https://github.com/tarkantemizoz/Cost-Sensitive-Learning`, accessed in 02.01.2021.

44. Powers, S., J. Qian, K. Jung, A. Schuler, N. Shah, T. Hastie and R. Tibshirani, "Some methods for heterogeneous treatment effect estimation in high-dimensions", *Statistics in Medicine*, Vol. 37, No. 11, pp. 1767–1787, 2017.

45. Bertsimas, D., J. Dunn and N. Mundru, "Optimal Prescriptive Trees", *INFORMS Journal on Optimization*, Vol. 1, p. ijoo.2018.0005, 2019.

46. Hastie, T., R. Tibshirani and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, Springer, 2 edn., 2009.

47. LeCun, Y., L. Bottou, G. B. Orr and K.-R. Müller, "Efficient BackProp.", G. Montavon, G. B. Orr and K.-R. Müller (Editors), *Neural Networks: Tricks of the Trade (2nd ed.)*, Vol. 7700 of *Lecture Notes in Computer Science*, pp. 9–48, Springer, 2012.