EXAMPLE-DEPENDENT COST-SENSITIVE GRADIENT BOOSTING MACHINES FOR CREDIT SCORING

by

İlker Kurtuluş

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2016

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 Computational Science and Engineering

Boğaziçi University

2022

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Assist. Prof. Mustafa Gökçe Baydoğan for providing guidance and sup ervision throughout my master thesis. Thanks to his continual support and valuable feedback, I had a chance to work on such an interesting and meaningful topic. I am also grateful to jury members, Prof. Savaş Dayanık and Assist. Prof. Ahmet Onur Durahim, for their valuable comments and insights that led me to improve my research.

I am thankful to my beloved wife, Dilek, for supporting, helping and motivating me to reach my goals throughout this challenging progress. I am tremendously thankful to my beloved parents, Nuran and Önder Mehmet, for their lifetime support and encouragement. I believe that my father would be so proud. I would like to express my gratitude also for my sister, Ceren and our lovely cat Pepe, for their emotional support during stressful days.

Finally, I would like to thank to my longtime friends Barış, Deniz and Emre for their advice and support throughout my study.

ABSTRACT

EXAMPLE-DEPENDENT COST-SENSITIVE GRADIENT BOOSTING MACHINES FOR CREDIT SCORING

Although most of machine learning algorithms try to minimize cost-insensitive losses, many real world applications require cost-sensitive approaches where misclassification costs among classes differ from each other. In addition to misclassification costs, examples in data sets may have nonidentical costs which is a case of example-dependent cost-sensitive learning. For example in credit scoring, mistakenly rejecting a good borrower and approving a bad client with financial distress result in different costs. Additionally, providing variety of credit amounts to applicants makes the credit scoring example-dependent. In other words, falsely approving 100M$ and 1M$ loans produce unequal costs. To overcome this problem, this thesis proposes an example-dependent cost-sensitive loss function. With the introduced loss function, cost sensitivity is handled during the learning process. This is achieved by changing the traditional loss function of Gradient Boosting Machines with the proposed one to make it *Example-Dependent Cost-Sensitive Gradient Boosting Machines*. The proposed algorithm is tested on two real world data sets that include credit amounts and synthetically generated data sets. The algorithm is compared with cost-insensitive learners, previously proposed example-dependent cost-sensitive classifiers that handles cost-sensitivity during learning, a post-processing method called *Thresholding* and a pre-processing method *Oversampling* to make cost-insensitive classifiers cost-sensitive. Results show that our method outperforms those four methods in terms of financial savings.

ÖZET

KREDİ SKORLAMA İÇİN ÖRNEK-BAĞIMLI MALİYET-DUYARLI GRADYAN ARTIRMA MAKİNELERİ

Birçok makine öğrenmesi algoritması maliyet duyarsız kayıpları azaltmaya çalışsa da, birçok gerçek dünya uygulaması yanlış sınıflandırmanın sınıflara bağlı olarak farklı maliyetler oluşturduğu, maliyet-duyarlı yöntemlerin kullanılmasını gerektirir. Yanlış sınıflandırma maliyetlerine ek olarak, veri setleri içindeki örnekler aynı olmayan maliyetlere sahip olabilir, bu da örnek-bağımlı maliyet-duyarlı öğrenme problemidir. Örneğin kredi skorlamada, yanlışlıkla reddedilen iyi bir müşteri ile onaylanan finansal durumu kötü olan bir müşterinin yaratacağı maliyetler farklıdır. Buna ek olarak, farklı miktarlarda kredilerin başvuranlara sağlanması, kredi skorlamayı örnek-bağımlı hale getirir. Diğer bir deyişle, 100M$'lık bir kredi ile 1M$'lık bir kredinin yaratacağı maliyetler eşit değildir. Bu problemi çözmek için, tezde örnek-bağımlı maliyet-duyarlı bir kayıp fonksiyonu öneriliyor. Önerilen kayıp fonksiyonu ile maliyet duyarlılık öğrenme sürecinde çözülüyor. Bu çözüme, Gradyan Artırma Makineleri'nin geleneksel kayıp fonksiyonunu, önerilen kayıp fonksiyonu ile değiştirerek ulaşıyoruz. Bu değişim ile Gradyan Artırma Makineleri'ni örnek-bağımlı maliyet-duyarlı hale getiriyoruz. Önerdiğimiz algoritmayı kredi miktarlarını içeren iki gerçek dünya veri setinde ve sentetik veri setlerinde deniyoruz. Algoritmayı, maliyet-duyarsız algoritmalarla, daha önce önerilen maliyet duyarlılığı öğrenme sürecinde halletmeye çalışan örnek-bağımlı maliyet-duyarlı sınıflandırma algoritmalarıyla, maliyet duyarsız algoritmaları maliyet-duyarlı hale getiren *Thresholding* isimli ön-işleme ve *Oversampling* isimli son-işleme yöntemleri ile karşılatıyoruz. Sonuçlar gösteriyor ki, finansal tasarruf açısından yöntemimiz bu dört yöntemden daha iyi çalışıyor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

α	Cost coefficient for false negative predictions
β	Cost coefficient for false positive predictions
$\mathcal{L}$	Loss function
c_i	Cost of an example
$\vec{c}$	Cost vector
c_{TP}	Cost of true positive
c_{FP}	Cost of false positive
c_{TN}	Cost of true negative
c_{FN}	Cost of false negative
y	Actual values
y_i	Actual value of an example
p	Predicted probability
p_i	Predicted probability of an example
$\hat{y}$	Predicted label
$\hat{y}_i$	Predicted label of an example
p^*	A cost-sensitive threshold
p_0	A cost-insensitive threshold
f	Learning function
$\hat{f}$	Estimated function
ϵ	Error
n	Number of samples
x	Feature set
λ	Learning rate
$F_m(x)$	Predicted log(odds) during boosting iteration m
$Res(f)$	Residual of a function f
$\mathbf{H}(f)$	Hessian of a function f
γ_{jm}	Output value of a node j at boosting iteration m

LIST OF ACRONYMS/ABBREVIATIONS

ANN	Artificial Neural Networks
AUC	Area Under ROC Curve
CM	Cost Matrix
CV	Cross Validation
DT	Decision Tree
EDCS_DT	Example-Dependent Cost-Sensitive Decision Tree
EDCS_GBM	Example-Dependent Cost-Sensitive GBM
EDCS_LR	Example-Dependent Cost-Sensitive Logistic Regression
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GBM	Gradient Boosting Machines
IG	Information Gain
KNN	K-Nearest Neighbours
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MSE	Mean Squared Error
QDA	Quadratic Discriminant Analysis
RF	Random Forests
ROC	The Receiver Operator Characteristics
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
TPR	True Positive Rate
UCI	University of California, Irvine

1. INTRODUCTION

In credit scoring, the main challenge is to distinguish customers that are likely to pay back from defaulted ones. Decision process to approve or reject a loan is called underwriting in financial institutions. Interest of academicians to predict financial outcome of a loan and model underwriting process continue for many years [1]. Recently, classification algorithms are being used in underwriting to enhance, stabilize and automatize lending process. The most typical approach is utilizing classification algorithms based on previous financial characteristics of customers such as delinquency, income and debt information. In binary setup, classification models are utilized to classify good (paid) and bad (defaulted) customers.

The most popular algorithms in credit scoring are Logistic Regression (LR) and Linear Discriminant Analysis (LDA) [1–3]. Additionally, K-Nearest Neighbours (KNN), Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Decision Trees (DTs) have been applied for credit scoring [4]. Decision Trees are widely used for credit scoring since their ability to learn nonlinear behaviour of data and generate easily understandable rules in term of business perspective [5]. In 2000s, ensemble methods were rising in the field of credit scoring therefore many researches [6–8] adopted ensemble learners such as Random Forests (RF) [9], AdaBoost [10] and Gradient Boosting Machines (GBM) [11] into credit risk assessment.

False predictions in classification models lead different type of costs based on actual values. For example, during heart disease classification, cost of predicting a human with disease as healthy may cause serious consequences such as death and illness. On the contrary, predicting an healthy person as diseased causes different outcome based on the treatment. Another example can be Covid-19 tests: Predicting a Covid-19 positive person as negative may increase spread of the virus. Also the person who is experiencing the disease without treatment may result in severe problems. Inversely, predicting a Covid-19 negative person as positive might cause only unnecessary quarantine which

has less cost compared to opposite situation. In credit scoring, the same concept applies such that realized costs differ based on actual values such as the debtor is a good client or not. Cost of the prediction for giving a loan to a bad client is the loan amount whereas cost of rejecting a good client is profit earned by the loan. In other words, for 10000$ loan, if the client do not pay the debt and the prediction was positive, which results in origination of the loan, the cost becomes 10000$ since the whole loan vanishes. However, in the case of rejecting a good client and missing the profit associated to interest rate, the cost becomes a portion of the loan based on the interest rate applied. If 8% interest rate is applied to the client, then the cost of the false prediction eventually turns out $0.08 \times 10000\$ = 800\$$ in that case.

In order to deal with such challenges in models where error types causes different costs, cost-sensitive learning approaches have been proposed over the years. In addition to cost caused by error types, instances may introduce different amount of costs. For example in credit scoring, cost of a 100$ loan and a 100M$ loan are different in terms of both approving a bad client and rejecting a good client. Therefore various loan amounts lead different costs which is a case of example-dependent cost-sensitive problem. Example-dependent cost-sensitive learning is an approach that aims to incorporate individual costs of each samples in a data set to statistical modelling [12].

Cost-sensitive learning can be divided into two main categories: Data level and algorithm level approaches [13]. Data level approaches aim to make classifiers cost-sensitive without changing the learning objective. There are two main categories of data level approaches such as *prior learning* and *post learning*. Prior learning approaches mostly focus on re-sampling of the input data based on cost differences of classes. Re-sampling techniques can be either oversampling or undersampling [14]. Oversampling synthetically generates observations belongs to minority class whereas undersampling is ignoring some of samples from majority class prior to learning. Even though those re-sampling approaches introduce cost sensitivity to imbalanced classes, they bring drawbacks such as loss of information [15].

In this research, we have proposed a novel loss function that takes into account not only cost of classes but also costs of examples. The loss function had been utilized in Gradient Boosting Machines in binary classification setup. Particularly, we proposed an Example-Dependent Cost-Sensitive Gradient Boosting Machines (in short EDCS_GBM) that aims to learn costs of examples and error types during generating an additive trees and updating cumulative predicted probabilities. The approach has been evaluated in 3 different data sets such as: a publicly available data set from UCI repository [16], a credit default data set from a major Turkish financial institution and finally a synthetic data set.

Chapter 2 demonstrates several type of classification and regression models, evaluation metrics for classification, error types and ensemble learners. In Chapter 3, previous works in the area of credit scoring in machine learning and cost-sensitive learning methods are described. Chapter 4 explains proposed EDCS_GBM approach in detail. Experimental setup, design, data sets and comprehensive results can be found in Chapter 5. And finally, discussion and conclusion of the approach are described in Chapter 6.

2. BACKGROUND

Since we deal with a specialized loss function that used in boosted regression trees, we want to divide the background section into three parts. Firstly, we will go through fundamentals of statistical learning by emphasizing loss functions, secondly we will explain ensemble of tree learners and finally, well known classification performance metrics and cost-sensitive metrics will be discussed in detail.

2.1. Statistical Learning

In supervised learning, the main objective is to learn targets or labels y by using input feature matrix X. A learned function, $\hat{f}$, minimizes error ϵ between real values y and predicted values $\hat{y}$:

$$y = f(X) + \epsilon$$
$$\hat{y} = \hat{f}(X).$$

Based on the target type, supervised learning approaches are considered under two main categories such as classification and regression. In the regression setup, the main objective is to learn continuous values of y while for the classification, it is to learn specific discrete classes such as whether a loan is defaulted or not, a patient has a disease or not. Predicting house prices, sales forecasting, weather forecasting can be examples for regression. In the classification setup the target is a finite subset while the target of the regression may take infinite number of choices.

The term loss function is usually used to define error between y_i's and $\hat{y}_i$'s. Minimization of total loss during a learning objective results in finding optimal parameters of $\hat{f}$. In regression, the most popular and common loss function is mean squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2. \tag{2.1}$$

In binary classification, in which objective is building an estimator to classify two classes, the most common loss function is log loss. Log loss is total sum of negative log likelihood of each predictions given target values:

$$\mathcal{L}(y, p) = -\sum_{i=1}^{n}(y_i \log p_i + (1 - y_i) \log(1 - p_i)). \tag{2.2}$$

In terms of error minimization and learning parameters, Decision Tree classifiers have a different approach compared to above loss functions. Variables and corresponding split values are calculated by a method called *split criterion*. There are two main split criteria: *Gini* [17] and *Entropy* [18].

Gini of a node is calculated for K classes where $P(y = k)$ is the probability distribution of a class k as follows:

$$Gini(node) = 1 - \sum_{k=1}^{K} P_k(y = k)^2. \tag{2.3}$$

Similarly, entropy of a node calculated by following equation:

$$Entropy(node) = -\sum_{k=1}^{K} P_k(y = k) \log_2 P_k(y = k).$$

To handle cost-sensitivity in terms of misclassification, class weights can be changed in both Equation (2.2) and Equation (2.3). This weighting can be achieved by using costs of different types of false predictions such that approving a bad borrower and rejecting a good client in credit scoring.

In order to select a candidate split, *Information Gain* (IG) is calculated by using Entropy and Gini as follows:

$$IG_{Gini}(node) = Gini(N_0) - \sum_{i} Gini(N_i)$$

$$IG_{Entropy}(node) = Entropy(N_0) - \sum_{i} Entropy(N_i),$$

where N_0 is the node before splitting and N_i's are nodes formed after splitting.

Information gain reflects impurity improvement of a split. In Decision Trees information gains are compared and a split that minimizes impurity at most is selected for partitioning the feature subset. It should be noted that Gini and Entropy are cost-insensitive measurements meaning that they assume different false predictions have same costs. In Decision Tree Regression, MSE in Equation (2.1) is one of the most widely used split criterion. A split that provides maximum MSE decrease is considered as the best split candidate.

2.2. Tree-Based Ensembles

Ensemble learning is concept that forms a strong learner by combining many base learners. This approach aims to improve bias in models and provide robustness in terms of predictions. Two of the most important methods of ensemble learning are *bagging* [19] and *boosting* [20]. In bagging, in other words bootstrap aggregation, a data set undergoes sampling without replacement multiple times to form smaller data sets. Those smaller data sets are trained with base learners in parallel and their final results aggregated by either majority voting or averaging. Finally, a strong model is formed by training base learners with different portions of the data set. However boosting training mechanism is different. In boosting, base learners use all samples of data set and form a strong learner in an additive manner. In other words, in each boosting iteration predictions of previous base learners is used for the current base learner in order to minimize a common loss function.

In Gradient Boosting Machines, each base learner is subjected to negative gradient of the loss function. The main idea is minimization of the loss function by forming an additive model with base learners and handling the optimization by a gradient descent method. One of the most important capabilities of GBM is that the algorithm can accept any differentiable function as a loss function [11]. This functionality has motivated us to propose cost-sensitive GBM by a cost-sensitive loss function. More details about GBM and the cost-sensitive function are available in Chapter 4.

2.3. Classification Performance Metrics

Minimization of the loss function, in other words finding the most optimum parameters of $\hat{f}$, occurs in a stage called *training*. In order to understand how the model behaves in unseen data, real target values (p) and model predictions ($\hat{p}$) of a *test* data set evaluated by *metrics*. Confusion matrix is matrix such that contains number of predictions among all classes and their actual values. The confusion matrix helps to understand overall performance of a model. For binary case, the classification matrix as follows:

Table 2.1. Confusion Matrix.

	Predicted Negative $\hat{y}_i = 0$	**Predicted Positive** $\hat{y}_i = 1$
Actual Negative $y_i = 0$	*True Negative (TN)*	*False Positive (FP)*
Actual Positive $y_i = 1$	*False Negative (FN)*	*True Positive (TP)*

For classification the most popular metrics are accuracy, f-score, precision, recall, and area under curve (AUC). True positive rate (TPR) is the same metric with recall which is measure for how does the model perform well by catching positive labels. False Positive Rate (FPR) is a metric shows how does the model perform bad catching positive labels. Receiver operating characteristic curve (ROC) is curve that has TPR in y-axis, FPR in x-axis. AUC score equals to area under ROC curve. Calculations of the metrics as shown in:

$$\begin{aligned}
\text{Accuracy} &= (TP + TN)/(TP + FP + FN + TN), \\
\text{Precision} &= TP/(TP + FP), \\
\text{Recall (FPR)} &= TP/(TP + FN), \\
\text{False Positive Rate} &= FP/(FP + TN), \\
F_\beta\text{-score} &= (1 + \beta^2)\frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}.
\end{aligned}$$

Selection β in F-score is based on how much recall is important to precision. In many cases it is selected as 1 in order to understand overall model performance on classes. False Positive is sometimes called *Type I Error* and similarly False Negative is called

Type II Error.

It can be quickly recognized that none of above metrics directly take into account cost of different error types such as false positives and false negatives. Elkan *et al.* proposed *cost matrix* in which costs of each prediction has been stated [14].

Table 2.2. Cost Matrix.

	Predicted Negative $\hat{y}_i = 0$	**Predicted Positive** $\hat{y}_i = 1$
Actual Negative $y_i = 0$	C_{TN}	C_{FP}
Actual Positive $y_i = 1$	C_{FN}	C_{TP}

In credit scoring, in case of labelling good clients as 1 and bad clients as 0, C_{FP} becomes default of a loan whereas C_{FN} is missing a good client in other words loosing the profit associated to the loan according to the cost matrix. C_{TP} and C_{TN} can be considered as 0 to focus realized costs or negative values in order to highlight opportunity costs and profits. However, the cost matrix in Table 2.2 does not contain cost of examples for example credit lines in credit scoring. It is only cost-sensitive in terms of error types not examples. This leads to construct an example-dependent cost-sensitive success metric.

Widely used classification performance metrics such as Accuracy, F-Score and AUC fall short in terms of business impact since they do not directly consider cost associated with examples and error types such as Type I and Type II. In order to compare different models in terms of total cost of a prediction objective, Bahnsen *et al.* proposed *Savings* metric such that [21]:

$$Savings(f(S)) = \frac{Cost(f(S)) - Cost_l(f(S))}{Cost_l(f(S))}, \tag{2.4}$$

where $Cost_l(f(S))$ and $f(S)$ are

$$Cost_l(f(S)) = \min\{Cost(f_0(S)), Cost(f_1(S))\},$$
$$Cost(f(S)) = \sum_{i}^{N} \Big(y_i(\hat{y}_i C_{TP_i} + (1-\hat{y}_i)C_{FN_i}) + (1-y_i)(\hat{y}_i C_{FP_i} + (1-\hat{y}_i)C_{TN_i}) \Big).$$

$Cost(f_0(S))$ is the total cost associated with predicting all examples as 0 and similarly $Cost(f_1(S))$ is the total cost associated with predicting all examples as 1. Selection of minimum of those two values provides the lowest cost without any learning procedure. Then interpretation of *Savings* becomes improvement of total cost compared to a baseline model.

3. RELATED WORK

Related work chapter has been divided into two parts: First, previous literature for credit scoring will be described and secondly past researches on cost-sensitive learning will be shown.

3.1. Credit Scoring

Throughout the years researches are introducing various methods for credit scoring in order to differentiate good and bad borrowers. Durand [22] was one of the first researchers to formulate credit score based on borrower properties. Altman [1] proposed a discriminant analysis by using financial ratios such as assets, capitals, earnings and equity to form Z index in order to predict whether a borrower will bankrupt or not. Following years, with the help of improvements in statistical learning and computational effectiveness, Linear Discriminant Analysis (LDA) and Logistic Regression (LR), K-Nearest Neighbours (KNN), Support Vector Machines (SVM), Quadratic Discriminant Analysis (QDA), Decision Trees (DT), Artificial Neural Networks (ANN), boosting and bagging models had been started to utilized.

Wiginton *et al.* proposed logit model by indicating LDA's short comings in terms of assumption of multivariate normal populations and limitation of discrete independent variables [2]. Steenackers focused on logistic regression to build a scoring system for personal loans by utilizing probability thresholds, so called *cut-off* points [3]. Henley *et al.* used KNN algorithm for credit assessment [23]. The focus on the research to take advantage of non parametric learning mechanism of KNN rather than parametric approaches such as logistic regression and LDA. Huang *et al.* showed that SVM based learners provide competitive performance compared to ANN, genetic programming, and Decision Tree classifiers [5]. Baesens *et al.* compared credit scoring models based on KNN, ANN, DT, SVM and least-squares SVM in two different data sets and found that SVMs and ANNs are slightly perform well in terms of accuracy and AUC score

compared to others and regarding to non linearity data sets, some classifiers might be perform well. They also suggested ensemble learning as a future work [4].

Wang *et al.* compared different ensemble learning methods (boosting, bagging and stacking) by using four base learners such as DT, SVM, LR and ANN to without utilizing ensemble learning on those base learners and show that in terms of average accuracy, type I and type II error, ensemble methods yield better performances [6]. Additionally, Brown *et al.* showed that in the presence of imbalanced class problem, ensemble models such that GBM and RF performs well compared to KNN, DT and QDA in terms of AUC [7].

All of the works up to this point use cost-insensitive metrics such as accuracy, F-score and AUC. In the next section, previously proposed cost-sensitive methods and metrics has been discussed.

3.2. Cost-Sensitive Learning

In previous years, in order to make learning problem cost-sensitive researches have proposed many approaches. While some of them focus on data level approaches such as sampling and weighting prior to learning and thresholding after learning, some of them aim to induce cost sensitivity during learning objective. Additionally, most of related work seeks to find a solution to only class costs associated with error types, yet there is very little work on costs dependent on examples. Also most of the researches provide comparisons of proposed methods by using cost-insensitive metrics such as F-Score, Accuracy, AUC and Type I, Type II errors.

Elkan *et al.* has introduced cost matrix in which true predictions may also have costs [14]. In order to make the learning problem cost-sensitive he have proposed several methods such as thresholding, re-balancing of samples including oversampling and undersampling, changing base rates. The work describes how to use changed base rates, which dependent on misclassification costs, for cost-insensitive estimators and apply

thresholding afterwards. The work focuses on data level approaches does not contain example-dependent costs. Another data level approach, in particular weighting, have been proposed by Kai Ming Ting *et al.* [24]. *An Instance Weighting* method introduced in the work in order to yield better performance based on number of high cost errors compared to C4.5 Decision Tree algorithm [25]. Zadrozny *et al.* aim to minimize costs of the prediction by *cost-proportionate weighting* which assigns different weights to samples in training set instead of changing the learning objective [26]. Sheng *et al.* have been suggested *Thresholding* in which they seek to find a probability threshold that minimizes total cost in test sets [27]. Similar to most of previous approaches, *Thresholding* does not consider costs of examples and proposes post learning data level approach.

McCarthy *et al.* showed that for large data sets include higher than 10.000 samples, approaches that are applied during learning in order to make objective cost-sensitive perform better than undersampling and oversampling methods [15]. Nayak *et al.* introduced *Miscost* which incorporates different costs related to Type I and Type II errors into the logit model during learning [28]. The method aims to minimize expected cost of classification and increase lenders profit in credit scoring. To make ANNs cost-sensitive, Kukar *et al.* proposed four methods to incorporate cost of classification into the back propagation [29]. Three of those methods (Cost-Sensitive Classification, Adapting the output of the network, adapting to learning) do not changing the learning problem which means they are data level approaches. Yet *Minimization of the misclassification costs* method modifies the error function so that proposes during learning approach to cost-sensitive learning and according to experiments yields the best results among all four methods. Fan *et al.* introduced *AdaCost* algorithm which makes AdaBoost algorithm cost-sensitive by applying weights derived from cost of misclassification in example level [30]. The weights are used in each boosting iteration yet learning functions of weak learners stay same. Another work for making Decision Trees cost-sensitive during learning done by Ling *et al.* [31]. They proposed *Decision Trees with Minimal Cost* which selecting attributes and splitting trees by minimizing cost rather than general splitting criteria such as entropy and gini. This work also considers only cost of

FP and FN by neglecting cost of examples. Sun *et al.* introduced *AdaC2* algorithm [13]. The algorithm converts AdaBoost algorithm to cost-sensitive by a weight updating strategy while taking into account imbalanced class problem.

Bahnsen *et al.* suggested *Example-Dependent Cost-Sensitive Logistic Regression* by modifying loss function that is minimized during the learning [21]. The loss function expects an example-dependent cost matrix in which cost of prediction varies based on both examples and type of predictions. They also applied the same loss function with the cost matrix to splitting criteria of Decision Trees [32]. And later on they proposed ensemble of Example-Dependent Cost-Sensitive Decision Trees including bagging, pasting, Random Forests and Random Patches [8]. Different than most of previous studies where cost-insensitive metrics are used, Bahnsen *et al.* proposed cost-sensitive *Savings* metric in order to compare different methods [21].

The mentioned ensemble of cost-sensitive Decision Trees study is limited to bagging, pasting, Random Forests and Random Patches, in other words it does not include boosting methods. Additionally, in those methods, credit lines which are real costs associated to loans are not directly used. Instead, credit lines were inferred using some parameters such as income and debt. Motivation of our study is making Gradient Boosting Machines example-dependent cost-sensitive by using directly credit lines of each applicants and applying different cost multipliers to each error types. In addition to bagging type ensembles and previously discussed AdaBoost methods, in this work, we focus on Gradient Boosting Machines by introducing example-dependent cost-sensitive loss function which is minimized during additive boosting iterations.

4. EXAMPLE-DEPENDENT COST-SENSITIVE GRADIENT BOOSTING MACHINES

As described previously, standard classification algorithms try to minimize cost-insensitive losses such as gini, entropy (for Decision Trees), log loss and misclassification costs. In order to make a learning objective cost-sensitive in terms of error types we have introduced α and β parameters to the log loss function. Additionally, to make standard log loss function example-dependent cost-sensitive, cost of instances, c_i's were combined to the loss function. By introducing c_i's, α and β into the learning objective of Gradient Boosting Machines, we propose *Example-Dependent Cost-Sensitive Gradient Boosting Machines* (EDCS_GBM). The loss function as follows:

$$\mathcal{L}(y, p) = \sum_{i=1}^{n} -c_i[y_i \alpha \log(p_i) + (1 - y_i)\beta \log(1 - p_i)], \tag{4.1}$$

where c_i is corresponding cost of an observation, α is cost coefficient of false negative, β is cost coefficient of false positive, y_i's are actual classes, p_i's are predicted probabilities and n is the number of samples.

For binary classification, the sigmoid function is used to convert log(odds) (γ), which is a prediction of each boosting iteration, to probabilities:

$$p = \sigma(\gamma) \tag{4.2}$$

$$= \frac{e^{\gamma}}{1 + e^{\gamma}}. \tag{4.3}$$

By using Equation (4.1) and Equation (4.2), $\mathcal{L}(y, \gamma)$ becomes:

$$\mathcal{L}(y, \gamma) = \sum_{i=1}^{n} -c_i[y_i \alpha \log(\frac{e^{\gamma_i}}{1 + e^{\gamma_i}}) + (1 - y_i)\beta \log(1 - \frac{e^{\gamma_i}}{1 + e^{\gamma_i}})].$$

The following pseudo-code summarizes steps of EDCS_GBM.

1: Initialize with $F_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma)$
2: **for** $m = 1$ to M **do**
3: $\quad r_{im} = -\left[\dfrac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$, i = 1 to N
4: $\quad$ Fit a regression tree to r_{im}'s. For each regression tree nodes i.e. regions R_{jm}:
5: $\quad$ **for** $j = 1$ to J **do**
6: $\quad\quad \gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{ij}} \mathcal{L}(y_i, F_{m-1}(x_i) + \gamma)$
7: $\quad\quad F_m(x) = F_{m-1}(x) + \lambda \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$
8: $\quad$ **end for**
9: $\quad p_m = \frac{e^{F_m(x)}}{1+e^{F_m(x)}}$ (p_f when $m = M$)
10: **end for**

Figure 4.1. Gradient Boosting Algorithm.

(i) To initialize GBM classifier, first we should find $F_0(x)$ that satisfies

$$F_0(x) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma).$$

To find $F_0(x)$, taking first derivative of the right-end and equate it to zero will be enough as shown in:

$$\frac{\partial}{\partial \gamma} \left(\sum_{i=1}^{n} \mathcal{L}(y_i, \gamma) \right) = 0. \tag{4.4}$$

We can find the initial p value that minimizes Equation (4.4) by using Equation (4.2) and

$$\sum_{i=1}^{n} -c_i \left[y_i(1-p)\alpha + (1-y_i)(-p\beta) \right] = 0.$$

Finally initial p becomes:

$$p = \frac{\sum_{i=1}^{n} c_i y_i \alpha}{\sum_{i=1}^{n} c_i [y_i(\alpha - \beta) + \beta]}.$$

In order to find $F_0(x)$, the inverse sigmoid with initial p is used:

$$F_0(x) = log(\frac{p}{1-p}).$$

(ii) From m = 1 to M:

(iii) Find *Residuals* by:

$$r_{im} = -\left[\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \tag{4.5}$$

$$= -\left[\frac{\partial[-c_i[y_i \log(\frac{e^{F(x_i)}}{1+e^{F(x_i)}})\alpha + (1-y_i)\log(1-\frac{e^{F(x_i)}}{1+e^{F(x_i)}})\beta]]}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \tag{4.6}$$

$$= c_i\left[y_i\alpha + \left(\frac{e^{F(x_i)}}{1+e^{F(x_i)}}\right)(y_i(\beta-\alpha)-\beta)\right]_{F(x)=F_{m-1}(x)}. \tag{4.7}$$

(iv) Fit regression trees to r_{im}'s and form decision tree nodes (regions) R_{jm} that are visualized in Figure 4.2.

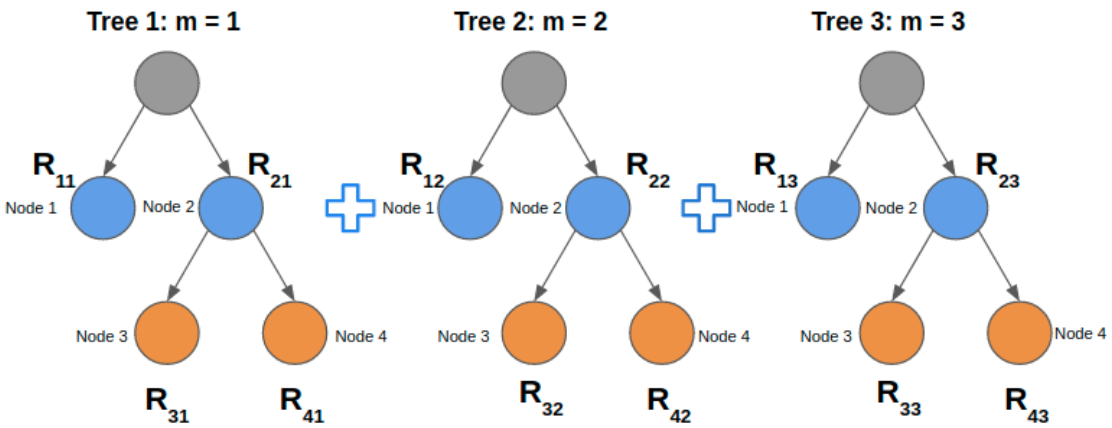

Figure 4.2. Additive Regression Trees of GBM and Their Nodes R_{jm}.

(v) In each boosting iteration m, and j^{th} node, namely regions R_{jm}, we need to find *log(odds)* that minimizes following function:

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} \mathcal{L}(y_i, F_{m-1}(x_i) + \gamma). \tag{4.8}$$

To find γ_{jm} that satisfies Equation (4.8), differentiation and equating to 0 will be sufficient since the loss function in the equation is differentiable. Before differentiation, second order Taylor polynomial expansion can be used as follows:

$$\mathcal{L}(y_i, F_{m-1}(x_i) + \gamma) \approx \mathcal{L}(y_i, F_{m-1}(x_i)) + \frac{d}{dF(x_i)}\left(\mathcal{L}(y_i, F_{m-1}(x_i))\right)\gamma + \frac{d^2}{dF(x_i)^2}\left(\mathcal{L}(y_i, F_{m-1}(x_i))\right)\gamma^2. \tag{4.9}$$

Differentiation of Equation (4.9) with respect to γ results in:

$$\frac{d}{d\gamma}\mathcal{L}(y_i, F_{m-1}(x_i) + \gamma) \approx \frac{d}{dF(x_i)}\left(\mathcal{L}(y_i, F_{m-1}(x_i))\right) + \frac{d^2}{dF(x_i)^2}\left(\mathcal{L}(y_i, F_{m-1}(x_i))\right)\gamma. \tag{4.10}$$

Right-end of Equation (4.10) should be zero to minimize Equation (4.8), therefore γ_{jm} becomes:

$$\gamma_{jm} = \frac{-\frac{d}{dF(x_i)}\left(\mathcal{L}(y_i, F_{m-1}(x_i))\right)}{\frac{d^2}{dF(x_i)^2}\left(\mathcal{L}(y_i, F_{m-1}(x_i))\right)\gamma}.$$

It can be easily observed that, γ_{jm} is ratio of negative gradient of a function to its hessian. Negative gradient is equivalent to *residual* as seen in Equation (4.5). Therefore, γ_{jm} can be also represented as:

$$\begin{aligned}\gamma_{jm} &= -\left[\frac{\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}}{\frac{\partial^2 \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)^2}}\right]_{F(x)=F_{m-1}(x)} \\ &= \frac{Res(y_i, \gamma_i)}{\mathbf{H}(y_i, \gamma_i)}.\end{aligned}$$

After calculating 1^{st} and 2^{nd} derivative of the loss function, γ_{jm} becomes:

$$\gamma_{jm} = \frac{\sum_{x_i \in R_{ij}} c_i \left[y_i\alpha + \left(\frac{e^{F(x_i)}}{1+e^{F(x_i)}}\right)(y_i(\beta-\alpha)-\beta)\right]}{-\sum_{x_i \in R_{ij}} c_i \left(\frac{e^{F(x_i)}}{1+e^{F(x_i)}}\right)\left(1-\frac{e^{F(x_i)}}{1+e^{F(x_i)}}\right)(y_i(\beta-\alpha)-\beta))}.$$

(vi) For the boosting iteration m, $F_m(x)$ is updated with a learning rate and γ_{jm} within node regions R_{jm} by the following equation:

$$F_m(x) = F_{m-1}(x) + \lambda \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm}).$$

(vii) And finally in order to obtain final predicted probabilities p_f after M number of boosting iterations, Equation (4.2) is applied to derive:

$$p_f = \frac{e^{F_M(x)}}{1 + e^{F_M(x)}}.$$

EDCS_GBM algorithm takes feature set X, actual values y and cost vector c that

contains cost of each example as inputs during training. Additionally, FN and FP coefficients α and β should be defined in parallel to related business objective and given to EDCS_GBM algorithm. After the training, for new predictions, the only input is feature set X which means cost of prediction samples are not used during the prediction phase.

5. COMPUTATIONAL EXPERIMENTS

In order to evaluate our proposed approach, we have used 3 different data sets: *Taiwan: Default of Credit Card Clients Data Set* provided in UCI Machine Learning Repository [16], *Loan Data Set* from a Turkish Financial Institution and finally *A Synthetic Data Set* which is generated by python programming language. For all data sets we have set α, cost of false negative, as 0.08 while β, cost of false positive, as 1. Selection of these parameters are under the assumption of fixed interest rate over loans.

In the loss function Equation (4.1), α and β parameters are fed to the model independent from the data, yet costs of examples, c_i's (credit lines in that case), should be given to the model as an input not a parameter. By providing those example costs to the model and using them during the learning make our approach example-dependent.

We compared EDCS_GBM algorithm with several methods that can be grouped in 4 main categories: Cost-insensitive learners, a post-processing method, a pre-processing method and example-dependent cost-sensitive learners that handle the sensitivity during learning similar to EDCS_GBM. Detailed information can be found in *Method* section. The whole approaches are tested with 10-Fold cross validation. For the synthetic data 10-Fold CV is applied 25 times with different random seed in order to minimize impact of randomness. In the *Results* section, we have reported 3 metric such that savings, realized savings and coverage for all data sets.

EDCS_GBM codes, training and hyperparameter tuning functions can be found in *csboosting* repository [33].

5.1. Data

In this research, to test proposed approach with previous researches, three different data sets are used in the concept of credit scoring:

- Taiwan Data Set
- Turkish Bank Data Set
- Synthetic Data Set

All of the data sets contain imbalanced class problem. Detailed properties are stated in Table 5.1. Default (unsuccessful payments) observations are labelled as 0 while successful payments are labelled as 1. For all credit models, ratio of α/β is selected as 0.08 which means cost of FP is 12.5 times of cost of FN. Credit amounts which are example costs for Taiwan Data Set and Turkish Bank Data Set are already exist in data sets. Example costs are given to the model by users of the algorithm. To avoid computational inefficiencies, example costs divided by maximum values of cost vectors. For the synthetic data set, in order to have a skewed example cost distribution, similar to real data sets such as Taiwan, following methodology is applied:

First of all, to create data sets, make classifier function of *scikit-learn* is used [34]. The function generates normally distributed informative features among clusters that are used as classes. Then the function generates redundant features which are linear combination of informative features. And lastly by adding some duplicated features and random noise, the feature sets are generated. In the experiments, number of features and classes are selected as 50 and 2 respectively. Then two different sample sizes are used: 1000 and 2000. To observe impact of class imbalance, three different class imbalance ratios (number of majority samples/number of all samples) are used: 0.7, 0.8 and 0.9. To create costs, mean and variance values of cost vector belongs to minority and majority classes of Taiwan data set are calculated separately. For each 6 (2 different sample sizes and 3 different class imbalanced ratios) synthetic data sets, by using mean and variance of the minority class costs, costs of minority class of synthetic data are produced. In this stage, *numpy* package is used to form normally distributed cost vectors [35]. Similarly costs of majority class of synthetic data are produced and then by combining those two costs vectors, final cost vectors (credit amounts) are created. In order to eliminate effect of randomness, the whole synthetic data sets are replicated 25 times for each learning objectives that are described in the next section.

Table 5.1. Properties of Taiwan and Turkish Bank Data Sets.

	Taiwan	**Turkish Bank**
Total Samples	30000	2562
Feature Count	22	59
Imbalanced Ratio (%)	77.8/22.2	90/10

5.2. Methods

In order to test EDCS_GBM and compare it to different methods and algorithms, these steps are followed:

- Comparison with cost-insensitive learners such as LR, DT, RF and GBM.
- Comparison with a post-processing approach: Thresholding is applied to LR, DT, RF and GBM [14].
- Comparison with a pre-processing approach: Oversampling is applied to LR, DT, RF and GBM [14].
- Comparison with estimators that aim to solve cost-sensitivity during learning: Example-Dependent Cost-Sensitive Decision Trees (EDCS_DT) and Example-Dependent Cost-Sensitive Logistic Regression (EDCS_LR) [21, 32].

5.2.1. Cost-insensitive Learners

To compare EDCS_GBM results with the most popular cost-insensitive learners, Decision Trees (DT), Random Forests (RF), Logistic Regression (LR) and Gradient Boosting Machines (GBM) have been chosen. An open source package, Scikit-learn is used for training [34]. In order to achieve fully cost-insensitivity so that appropriate comparisons, class weighting approaches are excluded in the algorithms.

5.2.2. Thresholding

In order to make cost-insensitive classifiers cost-sensitive, Elkan *et al.* proposed a post-processing approach: *Thresholding* [14]. According to the their study, based on cost of predictions, a cost-sensitive probability threshold should be:

$$p^* = \frac{c_{FP} - c_{TN}}{c_{FP} - c_{TN} + c_{FN} - c_{TP}}. \tag{5.1}$$

In our case, since we neglect costs of TP and TN, by integrating α and β to the Equation (5.1), p^* becomes:

$$\begin{aligned} p^* &= \frac{\beta}{\beta + \alpha} \\ &= 1/(1 + 0.08) \\ &= 0.9259. \end{aligned}$$

To assign predicted labels such as 0 and 1, in the *Thresholding* method, new threshold p^* is used instead of 0.5 for cost-insensitive algorithms: LR, DT, RF and GBM.

5.2.3. Oversampling

Elkan *et al.* also propose a pre-processing approach which is called *Oversampling* [14]. In this method, before the learning objective, class distributions are changed based on cost of false class predictions. Based on the study, minority class samples should be replicated by a factor

$$\frac{p^*}{1 - p^*} \frac{1 - p_0}{p_0}.$$

where p_0 is 0.5 for cost-insensitive estimators that we have used. Reminding that p^* is calculated in the last section, the replication factor becomes

$$\frac{\frac{\beta}{\beta+\alpha}}{1 - \frac{\beta}{\beta+\alpha}} \frac{1 - 0.5}{0.5} = \frac{\beta}{\alpha} = 12.5.$$

To implement this pre-processing method, minority samples are replicated 12.5 times for all data sets. It should be noted that *Thresholding* and *Oversampling* methods do not consider cost of examples, they only consider error types such that misclassification

costs based on classes.

5.2.4. During Learning Methods

As previously discussed, Bahnsen *et al.* proposes Example-Dependent Cost-Sensitive Logistic Regression (CS_LR) and Decision Trees (CS_DT) to handle cost-sensitivity during the learning phase [21, 32]. Similarly, our proposed EDCS_GBM method introduces costs associated with examples and classes during learning. *Costcla* package is used to experiment EDCS_DT and EDCS_LR algorithms [36].

For *Costcla* algorithms, a cost matrix, CM should be provided to the algorithm as an input. This matrix should contain all type of costs for each observations. As mentioned previously, we define costs of false positives as β and costs of false positives as α. These coefficients are multiplied with c_i's, credit amounts, to create cost matrix CM:

$$\begin{aligned} CM &= \begin{bmatrix} C_{FP} & C_{FN} & C_{TP} & C_{TN} \end{bmatrix} \\ &= \begin{bmatrix} \beta\vec{c} & \alpha\vec{c} & 0 & 0 \end{bmatrix}. \end{aligned}$$

For each data sets, cost matrices CM are calculated based on α, β and corresponding credit amounts (c_i's) of the data sets. It is important to note that the post-processing and the pre-processing methods that described in previous sections are not example-dependent different than EDCS_DT, EDCS_LR and EDCS_GBM algorithms.

5.3. Evaluation Methods

For evaluation we have used a cost-sensitive performance metric, a modified version of *Savings* in Equation (2.4), during cross validations and also evaluation of test sets. Motivations of the modification are adjusting the equation so that the best possible value becomes 1 and secondly using predicted probabilities in the evaluation metric.

Then the modified savings becomes:

$$Savings(y,p) = \frac{\min\{\mathcal{L}(y,p=0),\mathcal{L}(y,p=1)\} - \mathcal{L}(y,p)}{\min\{\mathcal{L}(y,p=0),\mathcal{L}(y,p=1)\}}.$$

The user should note that, *Savings* can take negative values since a model with worse performance compared to predicting all values as 0 or 1 may exist. Maximum value of *Savings* becomes 1 when loss of a particular prediction is 0.

In addition to *Savings*, two other metrics were also reported. These are *realized savings* and *coverage*. *Realized savings* is similar to *Savings*, the only difference is that assigned class labels are used instead of predicted probabilities. *Realized savings* can be interpreted as total savings after lending decisions are made in the concept of credit scoring. Therefore different probability thresholds may yield same *savings* but different *Realized savings* which can be calculated by:

$$Realized\_savings(y,\hat{y}) = \frac{\min\{\mathcal{L}(y,p=0),\mathcal{L}(y,p=1)\} - \mathcal{L}(y,\hat{y})}{\min\{\mathcal{L}(y,p=0),\mathcal{L}(y,p=1)\}}.$$

The other metric is *coverage* which is the ratio of total approved credit amount to total application amount in the data sets. In this work, credit amounts are example costs, therefore loan amounts in the data sets can be used directly to measure *Coverage*. The metric is pretty similar to "approval ratio in terms of loan amounts" in credit scoring. The motivation of the metric is to highlight importance of the total credit portfolio even the main objective is preserving defaulted loans. Formula of *Coverage* is as follows:

$$coverage = \frac{\sum_i^N c_i \hat{y}_i}{\sum_i^N c_i},$$

where c_i's are credit amounts (example costs) and $\hat{y}_i$'s are predicted labels.

5.4. Experimental Setup

For Taiwan and Turkish Bank Data Sets, two cross validations are used. For the outer 10-fold cross validation, for the inner 3 times repeated, 5-fold validation Validation is applied.

- 90% of samples are used in hyperparameter tuning. During hyperparameter tuning, we apply a 3 times repeated 5-fold stratified cross validation.
- The best hyperparameters are selected in terms of *Savings* metric.
- Same 90% of samples are fully trained with the best hyperparameters.
- The trained model is used to predict classes and probabilities of test samples (the rest 10%).
- Above steps are repeated 10 times for different folds.

For the synthetic data sets, two cross validations are also applied yet we apply a 3-fold stratified cross validation instead of a repeated cross validation. To eliminate effect of randomness, the steps that are applied for Taiwan and Turkish Bank data sets are repeated 25 times. Steps for training the synthetic data sets are as follows:

- A synthetic data set is created.
- 90% of samples are used in hyperparameter tuning. During hyperparameter tuning, we apply a 3-fold stratified cross validation.
- The best hyperparameters are selected in terms of *Savings* metric.
- Same 90% of samples are fully trained with the best hyperparameters.
- The trained model is used to predict classes and probabilities of test samples (the rest 10%).
- The tuning and the testing steps are repeated 10 times for different folds.
- Above steps are repeated 25 times by creating new data sets.

Model pre-processing steps such as scaling and oversampling are applied in both cross validations without any data leakages. During the hyper parameter tuning, an open source hyper parameter optimization framework have been used [37]. For Taiwan and Turkish Bank Data sets we select number of iterations as 50 in optuna while for synthetic data sets it has been selected 25 due to computational efficiency.

During training of Logistic Regression (LR) and Cost-Sensitive Logistic Regression (EDCS_LR) feature standardization is applied. Python3 is used as a programming

language. Machine learning library *scikit-learn* is applined during training of GBM, Decision Tree Classifier, Logistic Regression and Random Forest Classifier [34]. For EDCS_GBM, we use scikit-learn's Decision Tree Regression as base learner and prepare our custom GBM algorithm. Other cost-sensitive methods such as EDCS_LR and EDCS_DT *costcla* package is used [36].

Following hyper parameters are tuned and used for each corresponding algorithm during the cross validation:

EDCS_GBM:

- max_depth: (integer) 3 to 8
- learning_rate: (float) 0.01 to 0.3
- n_estimators: (integer) 100 to 500
- min_samples_leaf: (integer) 5 to 15

GBM:

- max_depth: (integer) 3 to 8
- learning_rate: (float) 0.01 to 0.3
- n_estimators: (integer) 100 to 500
- min_samples_leaf: (integer) 5 to 15

LR:

- penalty: (categorical) l1 and l2
- solver: (categorical) liblinear and saga
- tolerance: (float) 0.0001 to 0.01
- C: (float) 0.01 to 100
- fit_intercept: (categorical) True and False

DT:

- max_depth: (integer) 3 to 12
- min_samples_split: int, 2 to 20
- criterion: (categorical) gini and entropy
- min_samples_leaf: (integer) 2 to 20

RF:

- max_depth: (integer) 3 to 8
- min_samples_leaf: (integer) 5 to 15
- bootstrap: (categorical) True

EDCS_LR:

- solver: (categorical) bfgs and ga
- tolerance: (float) 0.0001 to 0.01
- C: (float) 0.01 to 100
- fit_intercept: (categorical) True and False

EDCS_DT:

- max_depth: (integer) 3 to 12
- min_samples_split: (integer) 2 to 20
- min_samples_leaf: (integer) 2 to 20

5.5. Results

We split results into two parts: Real world data sets and Synthetic data sets. Since EDCS_DT and EDCS_LR algorithms do not perform well and their run times are extremely high, we skip those approaches for the synthetic data sets.

5.5.1. Real World Data Sets

To compare different cost-sensitive methods, as discussed earlier, we follow the following steps to report results:

- EDCS_GBM vs Cost-insensitive learners
- EDCS_GBM vs Thresholding
- EDCS_GBM vs Oversampling
- EDCS_GBM vs previously proposed example-dependent cost-sensitive methods

5.5.1.1. EDCS_GBM vs Cost-insensitive learners. For both real world data sets, the best results in terms of savings are observed in EDCS_GBM. It should be noted that the whole algorithms are tuned based on savings. Yet still it can be observed in Figure 5.1, tree based algorithms (DT, GBM, RF) without any cost-sensitive improvements perform not so well in test folds. Moreover, they result in negative realized savings which means by using this estimators, predicted class labels generate worse realized savings compared to dummy predictions such that predict all as 0 or 1.

When we look at the coverages in Figure 5.1, we see that tree based methods have difficulties to assigning classes and predict nearly all samples as 1. Another interpretation of negative realized savings is that predicting all classes as 0, in other words rejecting the whole loans, is better than assigning all classes as 1 which is observed in the tree based methods.

Different than tree based methods, LR actually performs well in terms of savings and realized savings. EDCS_GBM still provides better savings results but when we look at Turkish Bank data set, in terms of realized savings LR performs better. However in terms of coverage which is associated with approval rate of credits, LR has lower coverage than EDCS_GBM. It can be quickly recognized that there is a trade-off between realized savings and coverage. Even though EDCS_GBM has higher savings, LR has higher realized savings but with less coverage. Based on business priorities, in this case

approval rate vs default rate of loans, by applying a thresholding for assigning class labels such trade-offs can be solved.

For Taiwan data set, the story is a bit different but the same trade-off is observed again. EDCS_GBM has higher realized savings but lower coverage compared to LR. Yet, since LR provides negative realized savings, it can be said that EDCS_GBM performs well compared to LR.

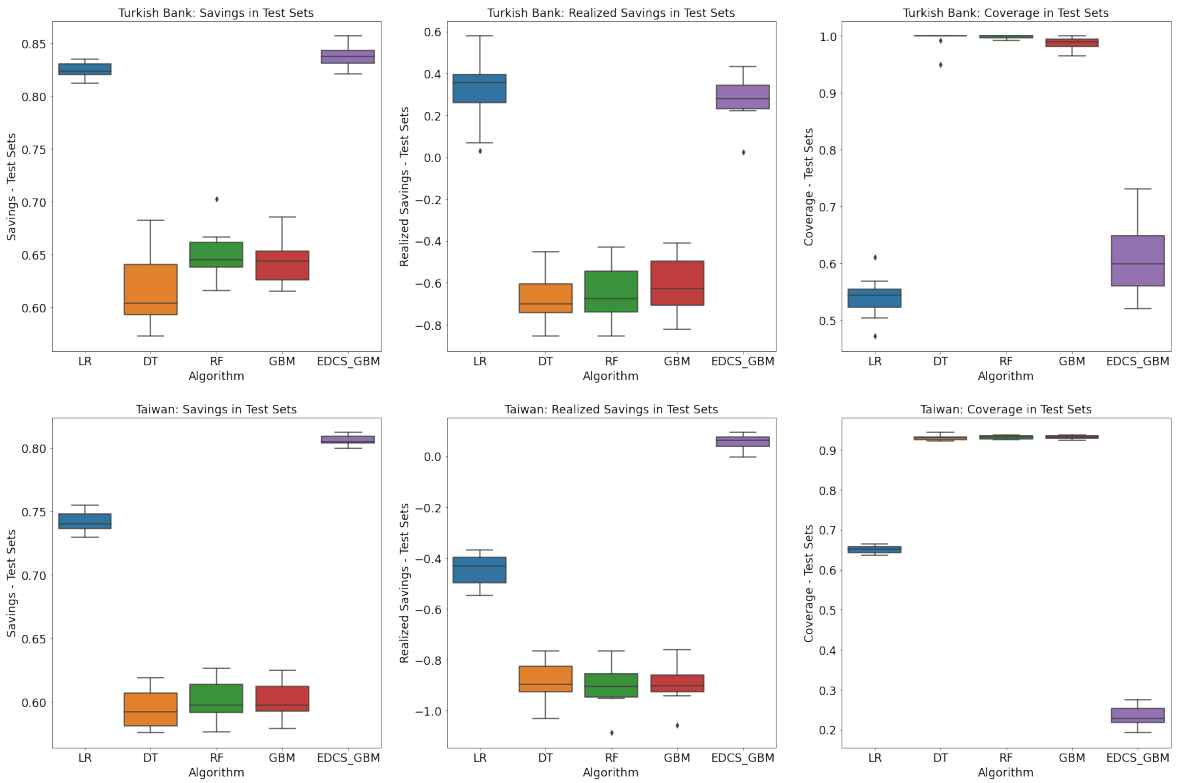

Figure 5.1. EDCS_GBM vs Cost-Insensitive Learners: Taiwan and Turkish Bank Test Results.

5.5.1.2. EDCS_GBM vs Thresholding. It is important to note that when *Thresholding* is applied to predicted probabilities, savings score does not change since savings uses predicted probabilities. Therefore in this section, compared to cost-insensitive estimators, savings results are same as the previous section. As in Figure 5.2, EDCS_GBM algorithm provides the best savings score.

When we look at realized savings, For Turkish Bank Data set, *Thresholding* improves tree based methods such that LR, GBM and RF yet worsens performance of LR. In the Figure 5.2, we see that LR starts to assign all classes as 0 with *Thresholding* therefore realized savings and coverage become 0 for LR.

In Turkish Bank data set, *Thresholding* improves tree based methods but still there is no major winner in terms of realized savings and coverage. EDCS_GBM and GBM with *Thresholding* dominate others but they compete with each other with very small differences with the trade-off explained before. In other words, EDCS_GBM has higher realized savings but lower coverage compared to GBM with *Thresholding*.

When we look at Taiwan data set results in Figure 5.2, LR and DT algorithms reject all loan applications (assigning class labels as 0 for all samples), therefore they both have 0 realized savings and coverage. For RF and GBM, *Thresholding* improves realized savings compared to cost-insensitive versions of the algorithms, however they are still outperformed by EDCS_GBM in terms of savings, realized savings and coverage.

To summarize, *Thresholding* improves some of algorithms for a data set but not for the other data set while it worsen successful algorithms when applied. Therefore we concluded that, similar to the cost-insensitive comparison, EDCS_GBM is a better choice compared to *Thresholding* in the presence of multiple data sets and different algorithms.

5.5.1.3. EDCS_GBM vs *Oversampling.* In Figure 5.3 it is observed that for both data sets, *Oversampling* improves tree based models in terms of savings and realized savings. Additionally *Oversampling* improves savings of LR for both data sets. Besides, the case of assigning all samples as 0 or 1 is not observed in *Oversampling* different than *Thresholding*.

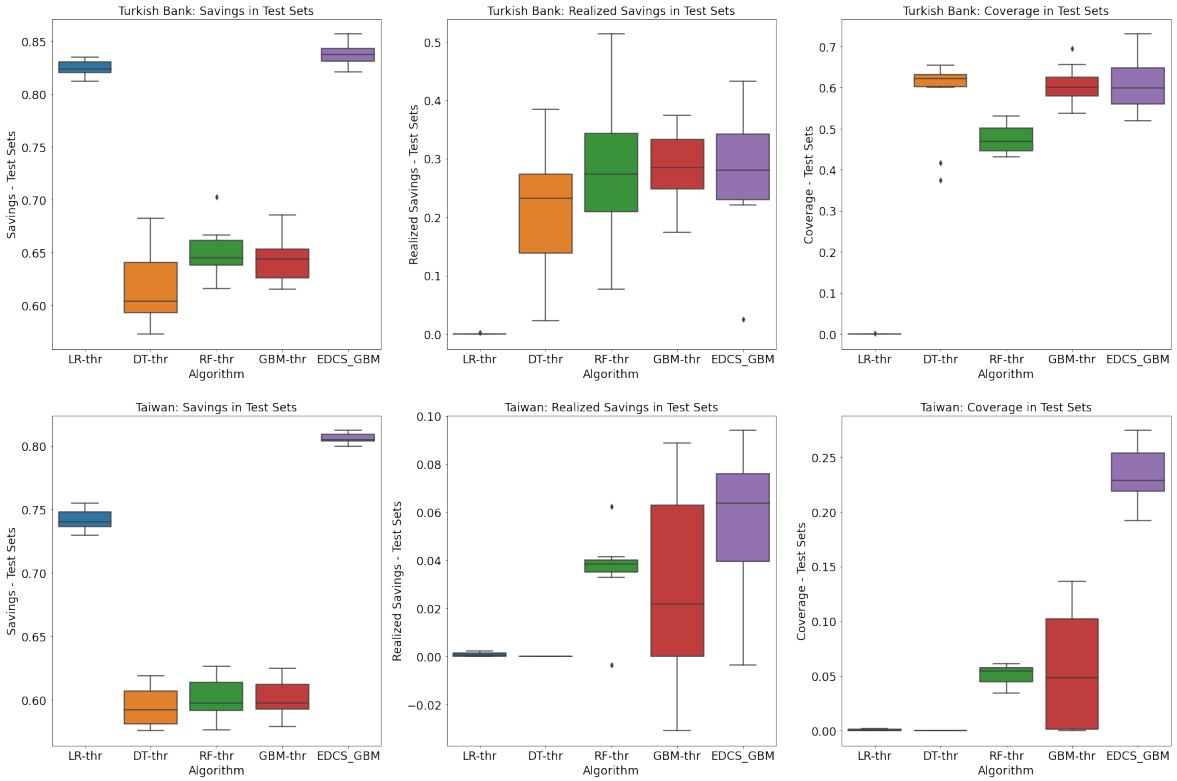

Figure 5.2. EDCS_GBM vs Thresholding: Taiwan and Turkish Bank Test Results.

For GBM, it can be seen that *Oversampling* yields better results compared to *Thresholding*. However this is not the case for DT. While *Oversampling* is better than *Thresholding* in Taiwan data set, it performs worse than *Thresholding* in Turkish Bank data set. RF with *Thresholding* is outperformed by RF with *Oversampling* in Taiwan data set. However these two approaches are pretty close to each other for Turkish Bank data set with the trade-off of coverage and realized savings. Even though *Oversampling* works better than *Thresholding* for LR, cost-insensitive LR provides better results for each data sets.

To summarize, while *Oversampling* improves some algorithms and provides better results compared to *Thresholding* or cost-insensitive methods, for different data sets and algorithms this improvement disappears. For Taiwan data sets, in terms of savings, EDCS_GBM and GBM with *Oversampling* perform the best scores which are pretty much close to each other. For the Turkish Bank data set, EDCS_GBM outperforms the whole oversampled models in terms of savings. When we look at realized savings in Figure 5.3, for both data sets, we can see coverage-realized savings trade-off between

EDCS_GBM and oversampled versions of RF and GBM. Since oversampled versions of specific algorithms provide better results in a data set and worse results in another one, one can actually consider EDCS_GBM as the best one. we can see coverage-realized savings trade-off between EDCS_GBM and oversampled versions of RF and GBM. Since oversampled versions of specific algorithms provide better results in a data set and worse results in another one, one can actually consider EDCS_GBM as the best one.

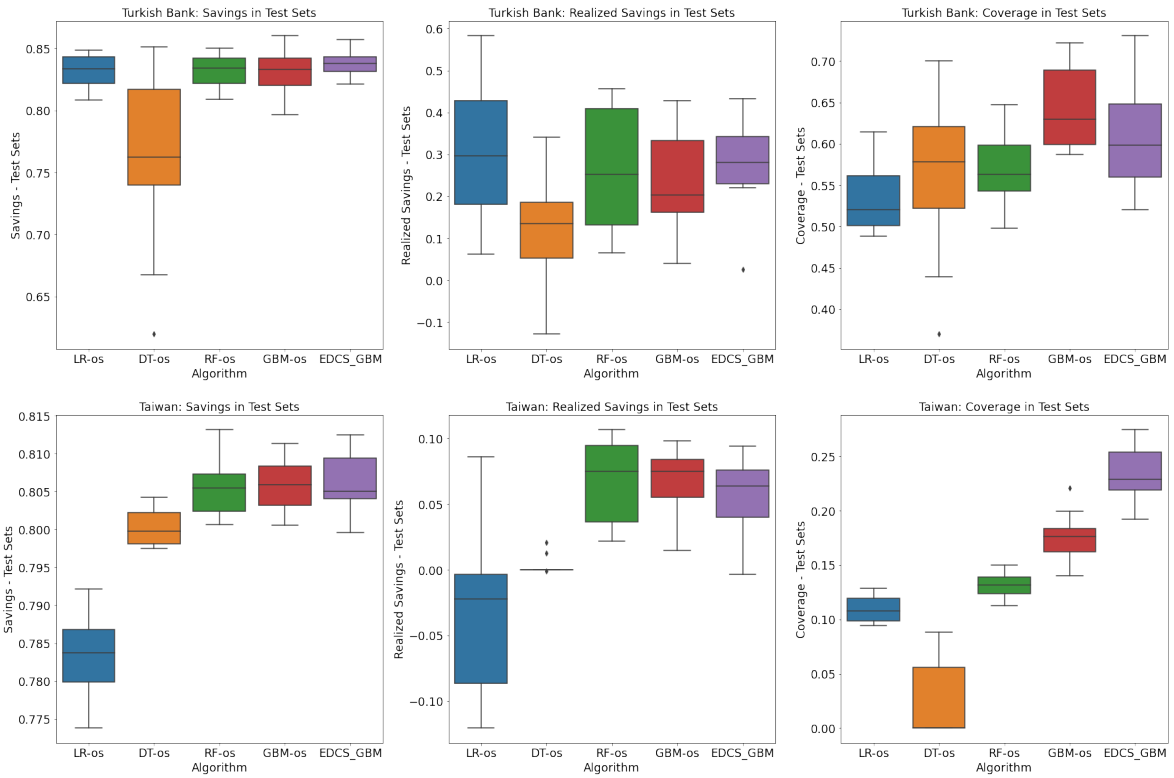

Figure 5.3. EDCS_GBM vs Oversampling: Taiwan and Turkish Bank Test Results.

5.5.1.4. EDCS_GBM vs EDCS_DT and EDCS_LR. When we look at Figure 5.4, it can be quickly recognized that EDCS_DT and EDCS_LR algorithms are assigned classes of all samples as 1. In other words, the algorithms approve the whole loan applications. It results in lower savings, negative actualized savings with 100% coverage for both data sets. In terms of profitability, in other words cost minimization, it can be clearly said that the proposed EDCS_GBM algorithm outperforms EDCS_DT and EDCS_LR algorithms.

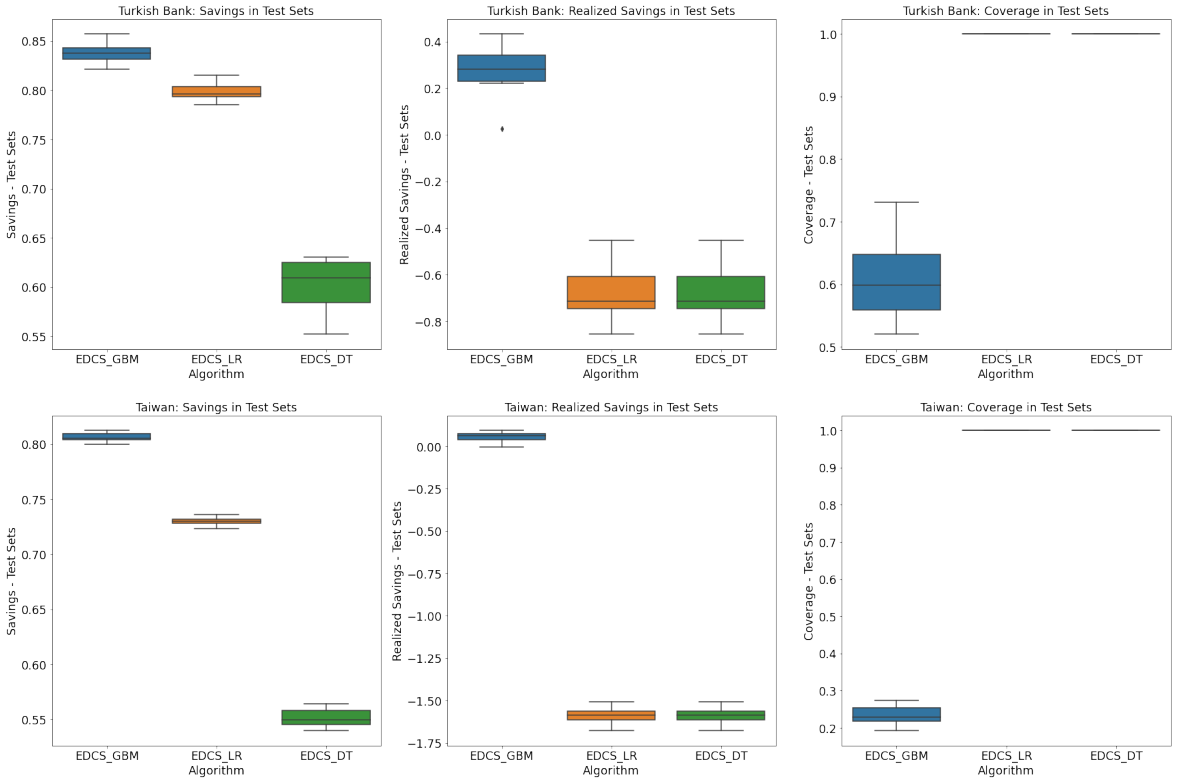

Figure 5.4. EDCS_GBM vs EDCS_DT and EDCS_LR: Taiwan and Turkish Bank Test Results.

To summarize, based on detailed results at Table 5.2 and Table 5.3, even EDCS_GBM ranks as 3^{rd} and 4^{th} in terms of realized savings, it has higher coverages compared to models with the highest realized savings. Because of that, it is impossible to say which model is the best based on realized savings. Additionally by applying a proper threshold that is aligned with modelling requirements, realized savings may increase while coverage decreases or vise versa. That's why we choose savings as the main success metric. In terms of savings metric EDCS_GBM yields the best performances and outperform the other 10 methods for both data sets.

Table 5.2. Taiwan Results.

Method	Algorithm	Savings - CV	Savings - Test	Realized Savings - Test	Coverage - Test
EDCS_GBM	**GBM**	**0.806 ± 0.001**	**0.806 ± 0.004**	0.055 ± 0.031	0.233 ± 0.028
Original	**GBM**	0.602 ± 0.001	0.602 ± 0.015	-0.895 ± 0.079	0.932 ± 0.005
Oversampling	**GBM**	**0.806 ± 0.001**	**0.806 ± 0.004**	**0.069 ± 0.025**	0.176 ± 0.023
Thresholding	**GBM**	0.602 ± 0.001	0.602 ± 0.015	0.029 ± 0.039	0.054 ± 0.054
EDCS_DT	**DT**	0.550 ± 0.001	0.551 ± 0.008	-1.594 ± 0.053	**1.000 ± 0.000**
Original	**DT**	0.593 ± 0.003	0.595 ± 0.016	-0.890 ± 0.081	0.930 ± 0.007
Oversampling	**DT**	0.800 ± 0.001	0.800 ± 0.003	0.003 ± 0.007	0.024 ± 0.039
Thresholding	**DT**	0.593 ± 0.003	0.595 ± 0.016	0.000 ± 0.000	0.000 ± 0.000
EDCS_LR	**LR**	0.730 ± 0.000	0.730 ± 0.004	-1.594 ± 0.053	**1.000 ± 0.000**
Original	**LR**	0.742 ± 0.001	0.742 ± 0.008	-0.445 ± 0.061	0.650 ± 0.010
Oversampling	**LR**	0.783 ± 0.001	0.783 ± 0.005	-0.034 ± 0.066	0.109 ± 0.012
Thresholding	**LR**	0.742 ± 0.001	0.742 ± 0.008	0.001 ± 0.001	0.001 ± 0.001
Original	**RF**	0.602 ± 0.001	0.601 ± 0.015	-0.905 ± 0.087	0.931 ± 0.005
Oversampling	**RF**	0.805 ± 0.000	0.805 ± 0.004	0.067 ± 0.031	0.131 ± 0.012
Thresholding	**RF**	0.602 ± 0.001	0.601 ± 0.015	0.036 ± 0.016	0.051 ± 0.010

Table 5.3. Turkish Bank Results.

Method	Algorithm	Savings - CV	Savings - Test	Realized Savings - Test	Coverage - Test
EDCS_GBM	**GBM**	**0.837 ± 0.003**	**0.837 ± 0.010**	0.280 ± 0.118	0.609 ± 0.067
Original	**GBM**	0.643 ± 0.006	0.643 ± 0.021	-0.619 ± 0.145	0.987 ± 0.011
Oversampling	**GBM**	0.831 ± 0.004	0.832 ± 0.019	0.230 ± 0.122	0.643 ± 0.052
Thresholding	**GBM**	0.643 ± 0.006	0.643 ± 0.021	0.288 ± 0.061	0.605 ± 0.047
EDCS_DT	**DT**	0.616 ± 0.007	0.600 ± 0.030	-0.677 ± 0.137	**1.000 ± 0.000**
Original	**DT**	0.613 ± 0.011	0.616 ± 0.034	-0.669 ± 0.130	0.994 ± 0.016
Oversampling	**DT**	0.757 ± 0.014	0.763 ± 0.075	0.117 ± 0.136	0.558 ± 0.097
Thresholding	**DT**	0.613 ± 0.011	0.616 ± 0.034	0.211 ± 0.105	0.580 ± 0.099
EDCS_LR	**LR**	0.798 ± 0.001	0.799 ± 0.010	-0.677 ± 0.137	**1.000 ± 0.000**
Original	**LR**	0.822 ± 0.002	0.824 ± 0.007	**0.331 ± 0.181**	0.539 ± 0.038
Oversampling	**LR**	0.828 ± 0.003	0.832 ± 0.014	0.306 ± 0.160	0.533 ± 0.043
Thresholding	**LR**	0.822 ± 0.002	0.824 ± 0.007	0.000 ± 0.001	0.000 ± 0.001
Original	**RF**	0.653 ± 0.005	0.65 ± 0.024	-0.655 ± 0.150	0.998 ± 0.003
Oversampling	**RF**	0.830 ± 0.001	0.831 ± 0.013	0.265 ± 0.151	0.570 ± 0.045
Thresholding	**RF**	0.653 ± 0.005	0.65 ± 0.024	0.285 ± 0.121	0.474 ± 0.036

5.5.2. Synthetic Data Sets

For synthetic data sets, the same methodology is applied to present comparable results. Plots are prepared in way that effect of number of samples and class imbalanced ratios are observable.

5.5.2.1. EDCS_GBM vs Cost-insensitive learners. Similar to real world data sets, the best result in terms of both savings and realized savings are generated by EDCS_GBM. Again, similar to real Taiwan and Turkish Bank data sets, LR outperforms other tree based methods in most of the cases. However it can be seen that, as class imbalanced ratio increases, savings performances of GBM increase faster than LR. This situation is observed for both sample sizes.

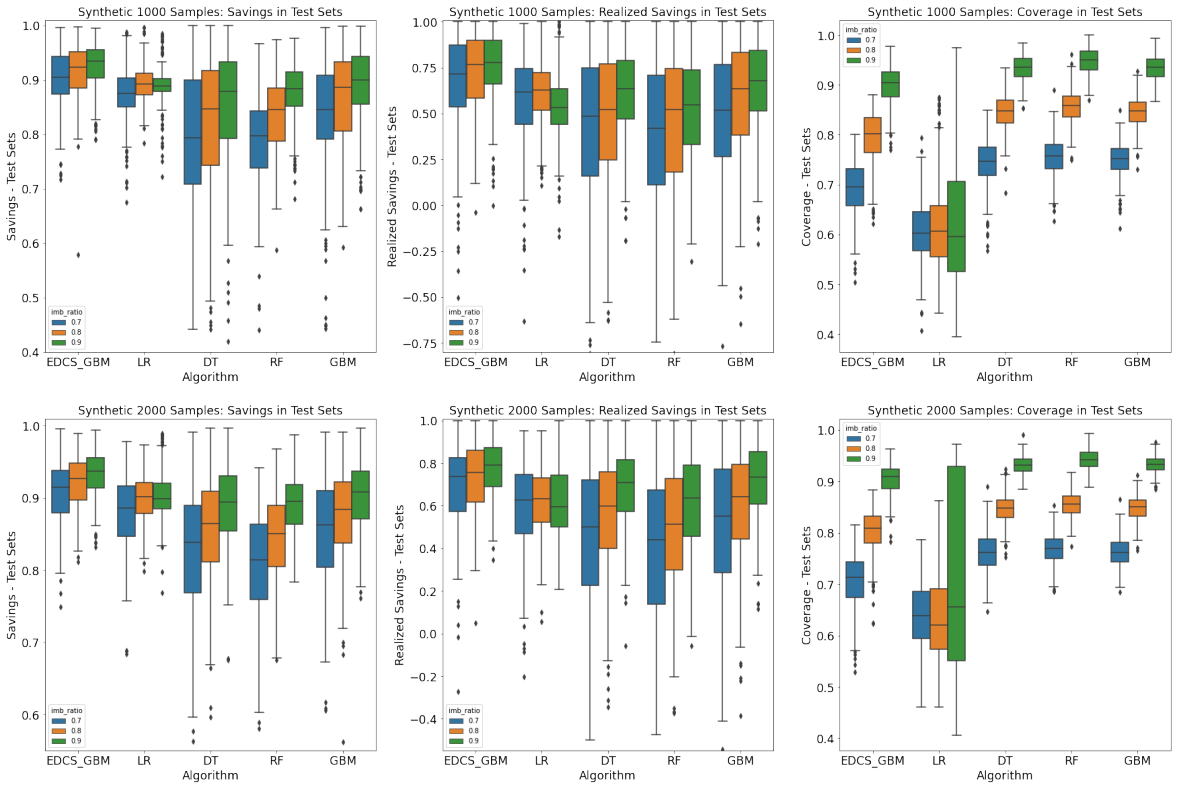

Figure 5.5. EDCS_GBM vs Cost-Insensitive Learners: Synthetic Data Test Results.

5.5.2.2. EDCS_GBM vs *Thresholding*. In *Thresholding* method, since savings score uses probabilities, savings results are same as cost-insensitive methods. The real impact

is observed in realized savings. Similar to cost-insensitive comparisons, as imbalanced class ratio and sample sizes increase, savings scores increase. *Thresholding* that makes cost-insensitive estimators cost-sensitive produces worse results than EDCS_GBM. In terms of realized savings, EDCS_GBM is still the best method for the whole 6 data sets.

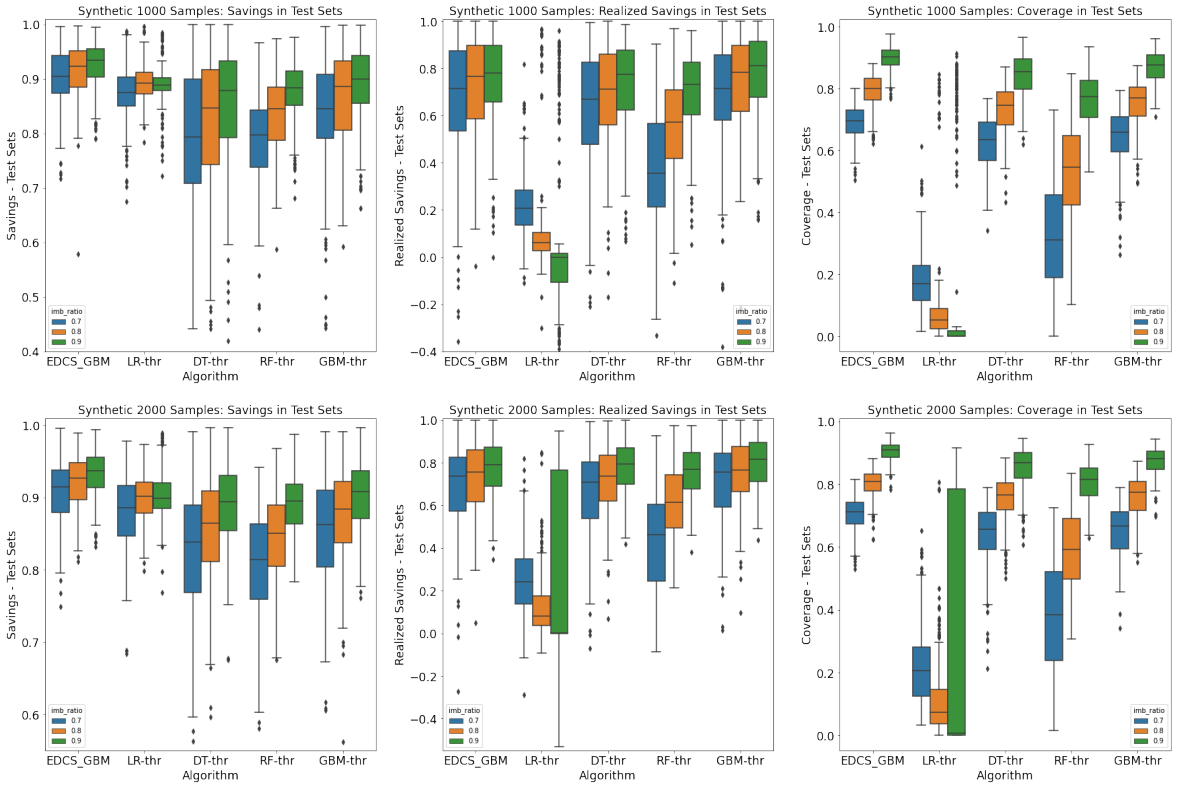

Figure 5.6. EDCS_GBM vs Thresholding: Synthetic Data Test Results.

5.5.2.3. EDCS_GBM vs *Oversampling*. *Oversampling* method performs well for GBM algorithm. When number of samples is 2000, in terms of savings, GBM performs better than EDCS_GBM algorithm. The savings differences are around 0.2%. While keeping number of samples as same and increasing class imbalanced ratio, savings score of EDCS_GBM increases faster than GBM-os.

When number of samples is halfed, EDCS_GBM outperforms all other methods. In addition to savings, for 0.7 and 0.8 imbalanced ratios, EDCS_GBM produces higher realized savings with higher coverage. For 2000 samples, even EDCS_GBM has lower realized savings, it produces higher coverages for the whole imbalanced ratios.

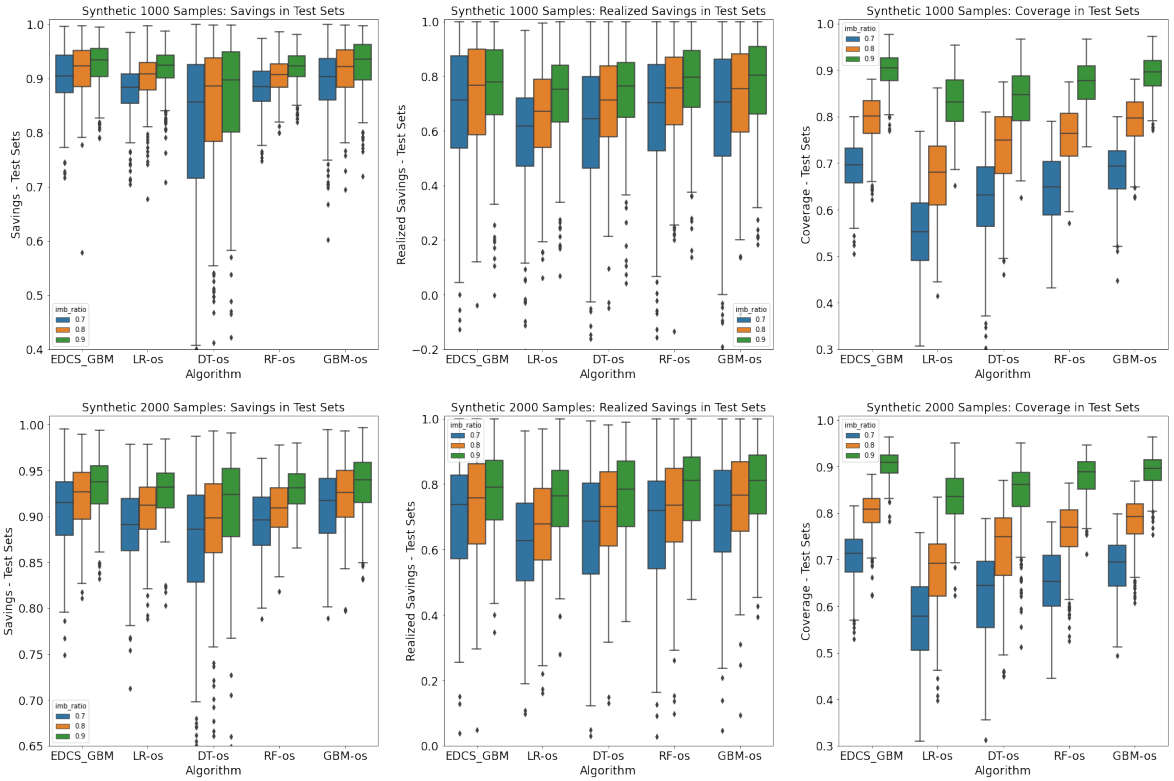

Figure 5.7. EDCS_GBM vs Oversampling: Synthetic Data Test Results.

5.5.2.4. EDCS_GBM vs EDCS_DT and EDCS_LR. It can be quickly recognized that EDCS_LR predicts the whole test sets as 1 for all 10 folds in 6 data sets. Since it has difficulties to differentiate classes, its savings and realized savings scores are the lowest. Even EDCS_DT does not predict all as 1, its savings scores are lower than EDCS_GBM. For the whole data sets EDCS_GBM is the best estimator in terms of savings. Additionally, EDCS_GBM has higher savings, realized savings and coverage than EDCS_DT except 0.9 class imbalanced ratio. In this exception, the difference of coverages are around 1% while EDCS_GBM has 30% more realized savings compared to EDCS_DT.

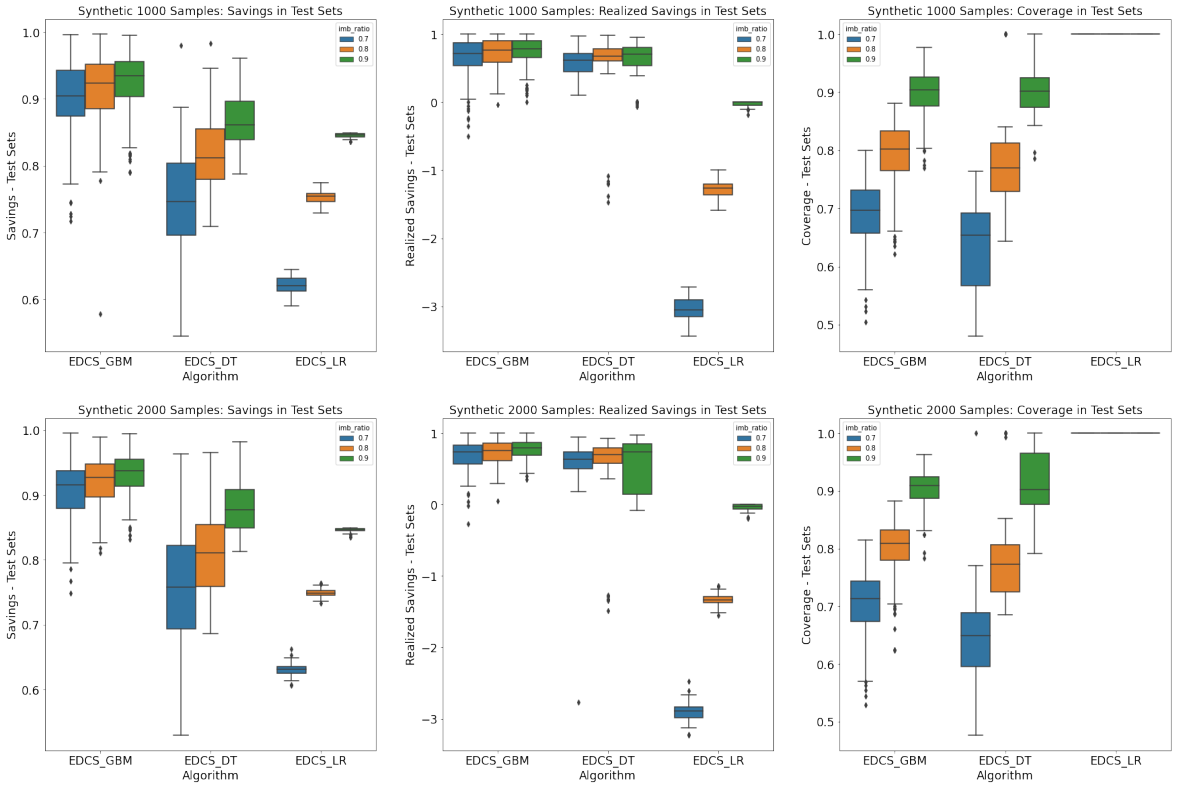

Figure 5.8. EDCS_GBM vs EDCS_DT and EDCS_LR: Synthetic Data Test Results.

6. CONCLUSION

In credit scoring, costs of misclassification and costs associated with varying loan amounts make the learning problem example-dependent cost-sensitive. Financial institutions generate profit based on interest rates when a loan successfully paid, yet their costs in case of delinquency much higher than the profit since interest income is only a small portion of the whole loans This is a case of cost-sensitive learning in terms of misclassification cost. Additionally, institutions provide variety of credit amounts to borrowers in other words costs associated with applicants (examples) also differ. In that sense credit scoring becomes an example-dependent cost-sensitive learning problem.

In this research, we propose an example-dependent cost-sensitive loss function and integrate it to Gradient Boosting Machines. This integration is performed by introducing example costs c_i, cost coefficients of false negative and false positive α, β to log loss function. Developed EDCS_GBM algorithm is used in three different credit scoring data sets. One of the data sets is publicly available *Taiwan: Default of credit card clients data set*, second one is a loan data set from a major Turkish financial institution and the last one is synthetically generated imbalanced class data sets. We compare EDCS_GBM with cost-insensitive classifiers such that LR, DT, RF and GBM, a post-processing method called *Thresholding* and a pre-processing method *Oversampling* to make those classifiers cost-sensitive. In addition to those 3 methods, we also apply previously proposed example-dependent cost-sensitive classifiers, EDCS_DT and EDCS_LR, that handle cost-sensitivity during the learning stage similar to our method. Since nature of the problem is cost-sensitive, we choose *Savings* metric as the main success criterion. In addition to *Savings* we also report realized savings and coverage metrics for each method.

For the real world data sets, Taiwan and Turkish Bank, we observe that EDCS_GBM outperforms other methods. Also for those data sets, EDCS_GBM provides stable performances which case cannot be seen in other methods since the other methods

work well for an estimator or a data set but do not perform well in another estimators or data sets. For synthetic data sets, EDCS_GBM outperforms all other methods in many cases, especially when there is low number of samples. When it is the second best performer with only 0.2% - 0.3% difference in savings, EDCS_GBM provides the highest coverages.

REFERENCES

1. Altman, E. I., "Financial Ratios, Discriminant Analysis and the Prediction of Corporate Bankruptcy", *The Journal of Finance*, Vol. 23, No. 4, pp. 589–609, 1968.

2. Wiginton, J. C., "A Note on the Comparison of Logit and Discriminant Models of Consumer Credit Behavior", *Journal of Financial and Quantitative Analysis*, Vol. 15, No. 3, pp. 757–770, 1980.

3. Steenackers, A. and M. Goovaerts, "A Credit Scoring Model for Personal Loans", *Insurance: Mathematics & Economics*, Vol. 8, No. 1, pp. 31–34, 1989.

4. Baesens, B., T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens and J. Vanthienen, "Benchmarking State-of-the-Art Classification Algorithms for Credit Scoring", *Journal of the Operational Research Society*, Vol. 54, No. 6, pp. 627–635, 2003.

5. Huang, C.-L., M.-C. Chen and C.-J. Wang, "Credit Scoring with a Data Mining Approach Based on Support Vector Machines", *Expert Systems With Applications*, Vol. 33, No. 4, pp. 847–856, 2007.

6. Wang, G., J. Hao, J. Ma and H. Jiang, "A Comparative Assessment of Ensemble Learning for Credit Scoring", *Expert Systems With Applications*, Vol. 38, No. 1, pp. 223–230, 2011.

7. Brown, I. and C. Mues, "An Experimental Comparison of Classification Algorithms for Imbalanced Credit Scoring Data Sets", *Expert Systems with Applications*, Vol. 39, No. 3, pp. 3446–3453, 2012.

8. Correa Bahnsen, A., D. Aouada and B. Ottersten, "Ensemble of Example-Dependent Cost-Sensitive Decision Trees", *arXiv E-Prints*, pp. arXiv–1505, 2015.

9. Breiman, L., "Random Forests", *Machine Learning*, Vol. 45, No. 1, pp. 5–32, 2001.

10. Freund, Y. and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", *Journal of Computer and System Sciences*, Vol. 55, No. 1, pp. 119–139, 1997.

11. Friedman, J. H., "Greedy Function Approximation: A Gradient Boosting Machine", *Annals of Statistics*, pp. 1189–1232, 2001.

12. Zadrozny, B. and C. Elkan, "Learning and Making Decisions When Costs and Probabilities Are Both Unknown", *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining of Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining*, pp. 204–213, 2001.

13. Sun, Y., M. S. Kamel, A. K. Wong and Y. Wang, "Cost-Sensitive Boosting for Classification of Imbalanced Data", *Pattern Recognition*, Vol. 40, No. 12, pp. 3358–3378, 2007.

14. Elkan, C., "The Foundations of Cost-Sensitive Learning", *International Joint Conference on Artificial Intelligence*, Vol. 17, pp. 973–978, Lawrence Erlbaum Associates Ltd, 2001.

15. McCarthy, K., B. Zabar and G. Weiss, "Does Cost-Sensitive Learning Beat Sampling for Classifying Rare Classes?", *Proceedings of the 1st International Workshop on Utility-based Data Mining*, pp. 69–77, 2005.

16. Dua, D. and C. Graff, *University of California, Irvine Machine Learning Repository*, 2017, `http://archive.ics.uci.edu/ml`, accessed in April 2022.

17. Breiman, L., J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Routledge, 2017.

18. Quinlan, J. R., "Induction of Decision Trees", *Machine Learning*, Vol. 1, No. 1, pp. 81–106, 1986.

19. Breiman, L., "Bagging Predictors", *Machine Learning*, Vol. 24, No. 2, pp. 123–140, 1996.

20. Schapire, R. E., "The Strength of Weak Learnability", *Machine Learning*, Vol. 5, No. 2, pp. 197–227, 1990.

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

22. Durand, D., *Risk Elements in Consumer Installment Financing*, National Bureau of Economic Research, New York, 1941.

23. Henley, W. and D. J. Hand, "A K-Nearest-Neighbour Classifier for Assessing Consumer Credit Risk", *Journal of the Royal Statistical Society: Series D (The Statistician)*, Vol. 45, No. 1, pp. 77–95, 1996.

24. Ting, K. M., "An Instance-Weighting Method to Induce Cost-Sensitive Trees", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 14, No. 3, pp. 659–665, 2002.

25. Salzberg, S. L., *C4. 5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann*, 1994.

26. 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.

27. Sheng, V. S. and C. X. Ling, "Thresholding for Making Classifiers Cost-Sensitive", *The Association for the Advancement of Artificial Intelligence*, Vol. 6, pp. 476–481, 2006.

28. Nayak, G. N. and C. G. Turvey, "Credit Risk Assessment and the Opportunity

Costs of Loan Misclassification", *Canadian Journal of Agricultural Economics/Revue canadienne d'agroeconomie*, Vol. 45, No. 3, pp. 285–299, 1997.

29. Kukar, M. and I. Kononenko, "Cost-Sensitive Learning with Neural Networks.", *ECAI*, Vol. 15, pp. 88–94, Citeseer, 1998.

30. Fan, W., S. J. Stolfo, J. Zhang and P. K. Chan, "AdaCost: Misclassification Cost-Sensitive Noosting", *International Conference on Machine Learning*, Vol. 99, pp. 97–105, Citeseer, 1999.

31. Ling, C. X., Q. Yang, J. Wang and S. Zhang, "Decision Trees With Minimal Costs", *Proceedings of the Twenty-First International Conference on Machine Learning*, p. 69, 2004.

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

33. Kurtulus, I., *Example-Dependent Cost-Sensitive Gradient Boosting Machines*, 2022, `https://github.com/ilkerkurtulus/csboosting`, accessed in April 2022.

34. 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", *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011.

35. Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, "Array Programming with NumPy", *Nature*, Vol.

585, No. 7825, pp. 357–362, 2020.

36. Bahnsen, C., *Costcla: Python Module For Cost-Sensitive Machine Learning Classification*, 2020, `https://pypi.org/project/costcla`, accessed in April 2022.

37. Akiba, T., S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-Generation Hyperparameter Optimization Framework", *Proceedings of the 25th International Conference on Knowledge Discovery and Data Mining of Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining*, 2019.