University of Luxembourg

Faculty of Science, Technology and Communication
Doctoral School of Computer Science and Computer Engineering
Interdisciplinary Centre for Security, Reliability and Trust

PhD dissertation

EXAMPLE-DEPENDENT COST-SENSITIVE CLASSIFICATION

APPLICATIONS IN FINANCIAL RISK MODELING AND MARKETING ANALYTICS

by Alejandro Correa Bahnsen

Advisor: Prof. Björn Ottersten
Advisor: Djamila Aouada
July 2015

[May 20, 2015 at 10:39 – version 0.3]

This dissertation has been submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science.

This version of the manuscript is pending the approval of the jury.

JURY MEMBERS

?????, Chairman
Professor, Université ?????

?????, Vice Chairman
Professor, Université ?????

Björn Ottersten, supervisor
Professor, University of Luxembourg

Djamila Aouada, supervisor
Research Scientist, University of Luxembourg

Bart De Moor
Professor, Katholieke Universiteit Leuven

ABSTRACT

Several real-world classification problems are example-dependent cost-sensitive in nature, where the costs due to misclassification vary between examples and not only within classes. However, standard classification methods do not take these costs into account, and assume a constant cost of misclassification errors. This approach is not realistic in many real-world applications. For example in credit card fraud detection, failing to detect a fraudulent transaction may have an economical impact from a few to thousands of Euros, depending on the particular transaction and card holder. In churn modeling, a model is used for predicting which customers are more likely to abandon a service provider. In this context, failing to identify a profitable or unprofitable churner has a significant different economic result. Similarly, in direct marketing, wrongly predicting that a customer will not accept an offer when in fact he will, may have different financial impact, as not all customers generate the same profit. Lastly, in credit scoring, accepting loans from bad customers does not have the same economical loss, since customers have different credit lines, therefore, different profit.

Accordingly, the goal of this thesis is to provide an in-depth analysis of example-dependent cost-sensitive classification. The first part, we give the general concepts of classification and cost-sensitive classification, and we present the cost-sensitive problem. The second part of this manuscript is dedicated to explain the particularities of the four real-world classification problems that are the focus of this thesis, in particular, credit card fraud detection, credit scoring, churn modeling and direct marketing. In general, we show why each of the applications is example-dependent cost-sensitive, and we elaborate a framework for the analysis of each problem. In the third part of this work, we introduce our previously proposed example-dependent cost-sensitive methods, namely, Bayes minimum risk, cost-sensitive logistic regression, cost-sensitive decision trees algorithm and ensemble of cost-sensitive decision trees. Moreover, we present the library *CostCla* that we develop as part of the thesis. This library is an open-source implementation of all the algorithms covered in this manuscript.

Finally, the results will show the importance of using the real example-dependent financial costs associated with real-world applications, since there are significant differences in the results when evaluating a model using a traditional cost-insensitive measure such as the accuracy or F1Score, than when using the savings, leading to the conclusion of the importance of using the real practical financial costs of each context.

To my wife Alejandra
it's a privilege to share my life with you.

ACKNOWLEDGMENTS

First and foremost, I would like to sincerely thank my advisor Prof. Dr. Björn Ottersten for the continuous support during my Ph.D study and research, for his motivation, enthusiasm, and for the freedom I was granted throughout these years.

My sincere thanks also goes to my co-advisor Dr. Djamila Aouada for her support, guidance, and encouragement not only from the research point of view, but for life in general.

Deepest gratitude are also due to all members of the jury for their interest in this work and for taking the time to evaluate this dissertation.

Special thanks go to the fraud management team at CETREL a SIX Company. In particular to Guy Weber and Philippe Davin, for all the time they invest in helping me understand all the particularities of the credit card fraud detection problem. Moreover, for their insights during the different steps of my research.

I would also like to convey thanks for supporting this Ph.D project to the AFR Grant Scheme (Aides la Formation-Recherche), managed by the National Research Fund of Luxembourg (FNR).

I cannot finish without saying how grateful I am with my wife Alejandra and son Pablo. They have always supported and encouraged me to do my best in all matters of life.

CONTENTS

NOTATIONS

$\mathcal{S}$	Set of examples
$\mathcal{S}_{train}$	Training set
$\mathcal{S}_{test}$	Testing set
$\mathcal{S}_0$	Set of the negative examples
$\mathcal{S}_1$	Set of the positive examples
$\mathbf{y}$	Class labels
$\mathbf{x}_i$	Feature vector of example i
k	Number of features
N	Number of examples in $\mathcal{S}$
N_0	Number of negative examples in $\mathcal{S}_0$
N_1	Number of positive examples in $\mathcal{S}_1$
y_i	Class label of example i
π_0	Percentage of negative examples in $\mathcal{S}$
π_1	Percentage of positive examples in $\mathcal{S}$
$f(\mathcal{S})$	A classification algorithm
$\mathbf{c}$	Predicted class labels
c_i	Predicted class for example i
$\hat{p}_i$	Predicted probability of positive for example i
t	Probability threshold
TP	Number of true positives examples
TN	Number of true negatives examples
FP	Number of false positives examples
FN	Number of false negatives examples
C_{FP}	Cost of a false positive
C_{FN}	Cost of a false negative
C_{TP_i}	Cost of a true positive for example i
C_{TN_i}	Cost of a true negative for example i
C_{FP_i}	Cost of a false positive for example i
C_{FN_i}	Cost of a false negative for example i
$\mathbf{x}_i^*$	Augmented feature vector of example i
$\mathbf{1}_c(z)$	Indicator function that takes the value of one if $z \in c$ and zero if $z \notin c$

$\|z\|$	Cardinality of z
b_i	Classification problem cost characteristic
μ_b	Mean of the classification cost characteristic
σ_b	Standard deviation of the classification cost characteristic
C_a	Administrative cost of contacting a customer
Amt_i	Amount of transaction i
t_p	Number of hours to aggregate transactions
x_i^{id}	Customer if of transaction i
x_i^{time}	Time of transaction i
int_r	Annual interest rate charged to a customer
L_{gd}	Credit scoring loss given default
r_i	Expected profit of customer i
CL_i	Credit line of customer i
l_i	Term of the loan of customer i
int_{cf}	Cost of funds of a financial institution
$debt_i$	Outstanding debt of customer i
q	Times income for calculating a credit line
γ	Probability of accepting an offer
CLV	Average customer lifetime value
CLV_i	Customer lifetime value of example i
C_{o_i}	Cost of making an offer to customer i
Int_i	Expected profit generated by customer i
θ	Parameters of the logistic regression
h_θ	Logistic regression hypothesis
$J(\theta)$	Logistic regression cost function
$J^c(\theta)$	Cost-sensitive logistic regression cost function
l^j	Splitting rule of a decision tree
I	Impurity measure of a decision tree
$Tree$	A decision tree
$Branch$	A branch of a decision tree

ACRONYMS

CS	Cost-sensitive classification
CI	Cost-insensitive classification
CPS	Cost-proportionate sampling
CST	Cost-sensitive training
LR	Logistic regression
DT	Decision tree
RF	Random forest
BMR	Bayes minimum risk
CSLR	Cost-sensitive logistic regression
CSDT	Cost-sensitive decision tree
ECSDT	Ensemble of cost-sensitive decision tree
CSB	Cost-sensitive bagging
CSP	Cost-sensitive pasting
CSRF	Cost-sensitive random forest
CSRP	Cost-sensitive random patches
t	Training
u	Under-sampling
r	Cost-proportionate rejection-sampling
o	Cost-proportionate over-sampling
mv	Majority voting
wv	Cost-sensitive weighted voting
s	Cost-sensitive stacking

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Classification, in the context of machine learning, deals with the problem of predicting the class of a set of examples given their features. Traditionally, classification methods aim at minimizing the misclassification of examples, in which an example is misclassified if the predicted class is different from the true class. Such a traditional framework assumes that all misclassification errors carry the same cost. This is not the case in many real-world applications. Methods that use different misclassification costs are known as cost-sensitive classifiers. Typical cost-sensitive approaches assume a constant cost for each type of error, in the sense that, the cost depends on the class and is the same among examples [Elkan, 2001; Kim et al., 2012].

This class-dependent approach is not realistic in many real-world applications. For example in credit card fraud detection, failing to detect a fraudulent transaction may have an economical impact from a few to thousands of Euros, depending on the particular transaction and card holder [Ngai et al., 2011]. In churn modeling, a model is used for predicting which customers are more likely to abandon a service provider. In this context, failing to identify a profitable or unprofitable churner has a significant different economic result [Verbraken et al., 2013]. Similarly, in direct marketing, wrongly predicting that a customer will not accept an offer when in fact he will, may have different financial impact, as not all customers generate the same profit [Zadrozny et al., 2003]. Lastly, in credit scoring, accepting loans from bad customers does not have the same economical loss, since customers have different credit lines, therefore, different profit [Verbraken et al., 2014].

Methods that use different misclassification costs are known as cost-sensitive classifiers. In particular we are interested in methods that are example-dependent cost-sensitive, in the sense that the costs vary among examples and not only among classes [Elkan, 2001]. However, the literature on example-dependent cost-sensitive methods is limited, mostly because there is a lack of publicly available datasets that fit the problem [Aodha and Brostow, 2013]. Example-dependent cost-sensitive classification methods can be grouped according to the step

where the costs are introduced into the system. Either the costs are introduced prior to the training of the algorithm, after the training or during training [Wang, 2013]. In Figure 1.1, the different algorithms are grouped according to the stage in a classification system where they are used.

The first set of methods that were proposed to deal with cost-sensitivity consist in re-weighting the training examples based on their costs, either by cost-proportionate rejection-sampling [Zadrozny et al., 2003], or cost-proportionate over-sampling [Elkan, 2001]. The rejection-sampling approach consists in selecting a random subset by randomly selecting examples from a training set, and accepting each example with probability equal to the normalized misclassification cost of the example. On the other hand, the over-sampling method consists in creating a new set, by making n copies of each example, where n is related to the normalized misclassification cost of the example. In [Correa Bahnsen et al., 2013, 2014b], we proposed a direct cost approach to make the classification decision based on the expected costs. This method is called Bayes minimum risk, and has been successfully applied to detect credit card fraud. The method consists in quantifying tradeoffs between various decisions using probabilities and the costs that accompany such decisions.

Moreover, in [Correa Bahnsen et al., 2014a], we proposed a new cost-sensitive logistic regression. The method consists in introducing example-dependent costs into a logistic regression, by changing the objective function of the model to one that is cost-sensitive. Then, in [Correa Bahnsen et al., 2015a], we proposed a new example-dependent cost-sensitive decision tree. The method is based on a new splitting criteria which is cost-sensitive, used during the tree construction. Then, after the tree is fully grown, it is pruned by using a cost-based pruning criteria. Lastly, in [Correa Bahnsen et al., 2015c], we proposed a ensemble of cost-sensitive decision tree framework. This is an extension of the previously proposed methods, in which the advantages of ensemble learning are used in order to have a more robust model.

We evaluate the different methods using five different databases from four real-world problems. In particular, credit card fraud detection, churn modeling, credit scoring and direct marketing. The results show that the proposed method outperforms state-of-the-art example-dependent cost-sensitive methods. Furthermore, our source code, as used for the experiments,

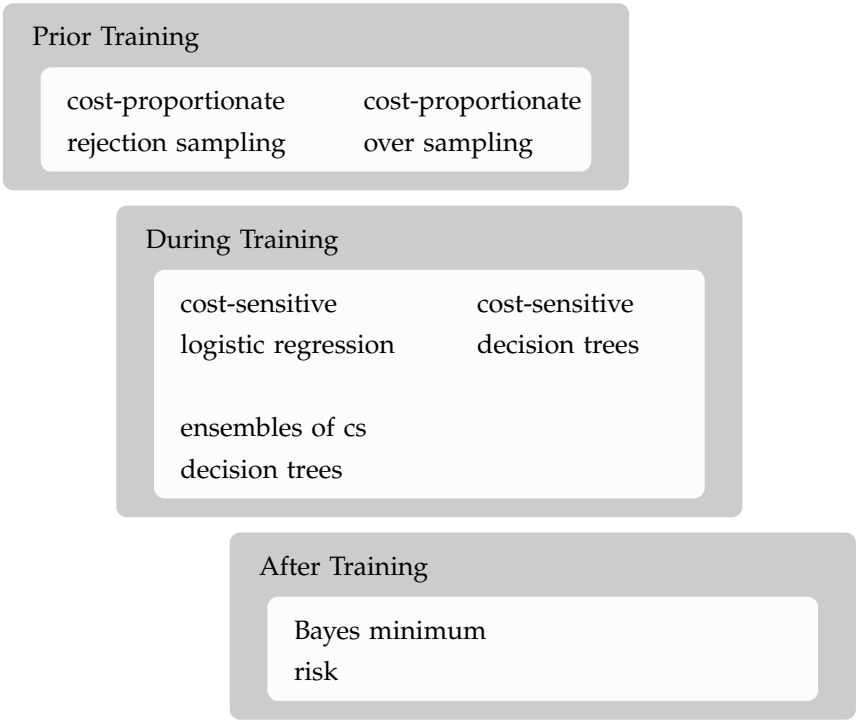


Figure 1.1: Different example-dependent cost-sensitive algorithms grouped according to the stage in a classification system where they are used.

is publicly available as part of the *CostSensitiveClassification*[1] library.

The results show that by taking into account the real financial costs of the different real-world applications, our proposed example-dependent cost-sensitive decision tree is a better choice for these and many other applications. This is because, our algorithm is focusing on solving the actual business problems, and not proxies as standard classification models do. We foresee that our approach should open the door to developing more business focused algorithms, and that ultimately, the use of the actual financial costs during training will become a common practice. Moreover, by creating many different decision trees and combining them using a cost-sensitive ensemble framework, the best results are found.

Finally, our results show the importance of using the real example-dependent financial costs associated with real-world applications, since there are significant differences in the results when evaluating a model using a traditional cost-insensitive measure such as the accuracy or F1Score, than when using the

1 https://github.com/albahnsen/CostSensitiveClassification

savings, leading to the conclusion of the importance of using the real practical financial costs of each context.

1.1 OUTLINE AND CONTRIBUTIONS

Part I of this manuscript is focused in giving the general concepts of classification and cost-sensitive classification. In particular, in Chapter 2, we give a self-contain background on classification. Then, in Chapter 3, we present the cost-sensitive problem. Moreover, in this chapter, we define what is the different between cost-insensitive, class-dependent cost-sensitive and example-dependent cost-sensitive classification problems. Lastly, we give an introduction of the different evaluation measures that are used throughout this thesis.

Part II is dedicated to explain the particularities of the four real-world classification problems that are the focus of this thesis, in particular, credit card fraud detection, credit scoring, churn modeling and direct marketing. In general, we show why each of the applications is example-dependent cost-sensitive, and we elaborate a framework for the analysis of each problem. The part is organized in two chapters. First, in Chapter 4, we discussed the applications within financial risk management, lastly, in Chapter 5, the marketing analytics applications.

Part III is focused in introducing our proposed example-dependent cost-sensitive methods. First, in Chapter 6, we present our Bayes minimum risk method. Then, in Chapter 7, we introduce our method cost-sensitive logistic regression. Afterwards, in Chapter 8, we show and discuss our previously proposed cost-sensitive decision trees algorithm. Additionally, in Chapter 9, we explain our proposed framework for ensembles of cost-sensitive decision trees. Lastly, in the Appendix A, we present the library *CostCla* that we develop as part of the thesis. This library is an open-source implementation of all the algorithms covered in this manuscript. [TODO] Djamila, is Appendix A named here? before conclusions?

Chapter 10 concludes the thesis, and elaborates on possible lines for future research.

1.2 PUBLICATIONS

This dissertation summarizes several contributions to the field of example-dependent cost-sensitive machine learning. Publications from this work include:

- [Correa Bahnsen et al., 2013] *Cost Sensitive Credit Card Fraud Detection Using Bayes Minimum Risk*, Alejandro Correa Bahnsen, Aleksandar Stojanovic, Djamila Aouada and Björn Ottersten. In Proceedings of IEEE International Conference on Machine Learning and Applications, 2013.

- [Correa Bahnsen et al., 2014b] *Improving Credit Card Fraud Detection with Calibrated Probabilities*, Alejandro Correa Bahnsen, Aleksandar Stojanovic, Djamila Aouada and Björn Ottersten. In Proceedings of SIAM International Conference on Data Mining, 2014.

- [Correa Bahnsen et al., 2014a] *Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring*, Alejandro Correa Bahnsen, Djamila Aouada and Björn Ottersten. In Proceedings of IEEE International Conference on Machine Learning and Applications, 2014.

- [Correa Bahnsen et al., 2015a] *Example-Dependent Cost-Sensitive Decision Trees*, Alejandro Correa Bahnsen, Djamila Aouada and Björn Ottersten. In Expert Systems with Applications, in press, 2015.

- [Correa Bahnsen et al., 2015b] *A novel cost-sensitive framework for customer churn predictive modeling*, Alejandro Correa Bahnsen, Djamila Aouada and Björn Ottersten. In Decision Analytics, in press, 2015.

- [Correa Bahnsen et al., 2015c] *Ensembles of Example-Dependent Cost-Sensitive Decision Trees*, Alejandro Correa Bahnsen, Djamila Aouada and Björn Ottersten. In IEEE Transactions on Knowledge and Data Engineering, 2015.

- [Correa Bahnsen et al., 2015d] *Feature Engineering in Credit Card Fraud Detection*, Alejandro Correa Bahnsen, Djamila Aouada and Björn Ottersten. In Expert Systems with Applications, 2015.

Part I

EXAMPLE-DEPENDENT COST-SENSITIVE CLASSIFICATION

2

BASICS ON CLASSIFICATION

Outline

In this chapter, we introduce classifications models from a machine learning perspective. First, in Section 2.1, we give a self contain introduction to classification, including the most-common algorithms, and the main applications of classification models. Then, in Section 2.2, we present the different evaluation measures that are normally used for analyzing the performance of classification methods.

2.1 INTRODUCTION

In machine learning, classification refers to the attempt of identifying to which of a set of classes a new example belongs, based on learning from examples whose class membership is known. The most important point about classification is that for each example only one know class is possible, making this a discrete problem.

A classification, task begins with a training set in which the class of a set of examples is know. For example, a classification model that predicts credit card fraud, is developed by analyzing many observed credit transactions over a period of time. The class in this case is a variable which indicates for each example whether or not the transaction was o not a fraud. Also, the predictors or features, are the transaction attributes like place, amount and time of the transaction.

Then, during the training process, a classification algorithm finds the patterns and relationships between the values of the features and the values of the target class. Different algorithms use different methods and techniques to estimate the relationships. Afterwards, these relationships are summarized in a model that is able to make predictions on new sets of data.

Formally, a classification algorithm deals with the problem of predicting the class y_i of a set $\mathcal{S}$ of examples or instances, given their k features $\mathbf{x}_i \in \mathbb{R}^k$. The objective is to construct a function $f(\mathcal{S})$ that makes a prediction c_i of the class of each of the N examples using its feature vector $\mathbf{x}_i$. Moreover, some algorithms

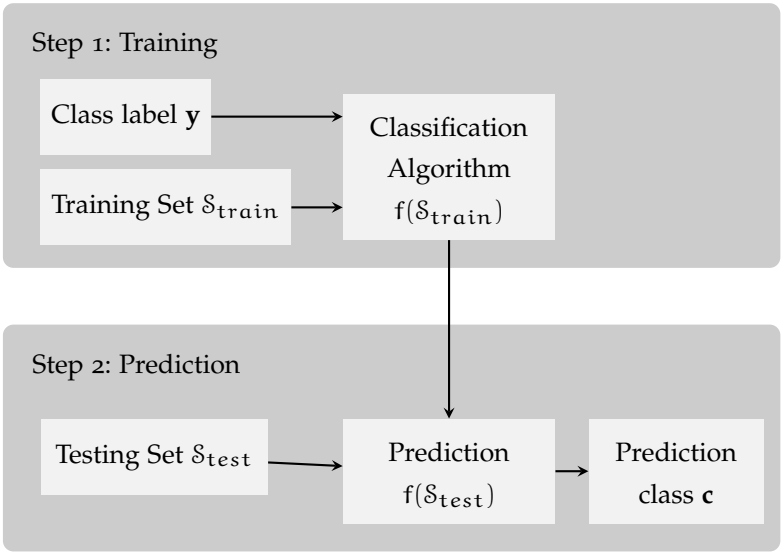


Figure 2.1: Classification process

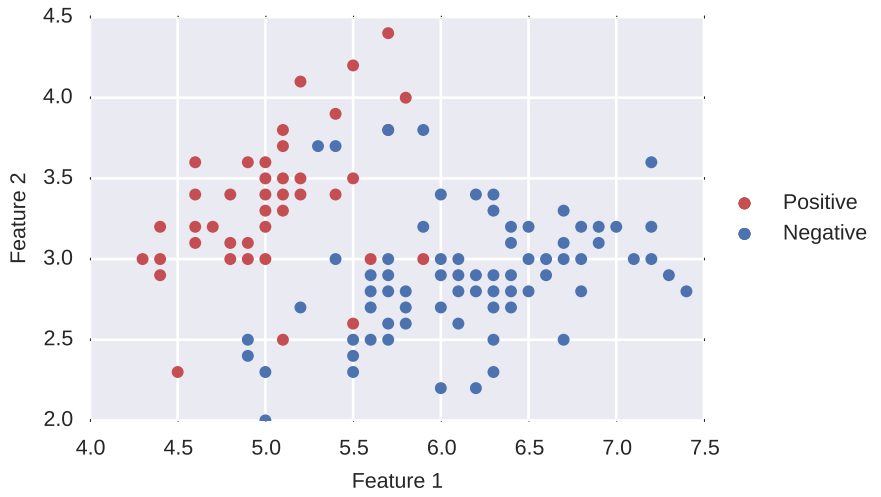
allows to not only know the prediction, but also the confidence of it, in the form of the probability of belonging to the positive class $\hat{p}_i$. The way for going from $\hat{p}_i$ to c_i, is simply by defining a probability threshold t, and applying the following formula

$$c_i = \begin{cases} 0 & \text{if } \hat{p}_i \leqslant t \\ 1 & \text{otherwise,} \end{cases} \tag{2.1}$$

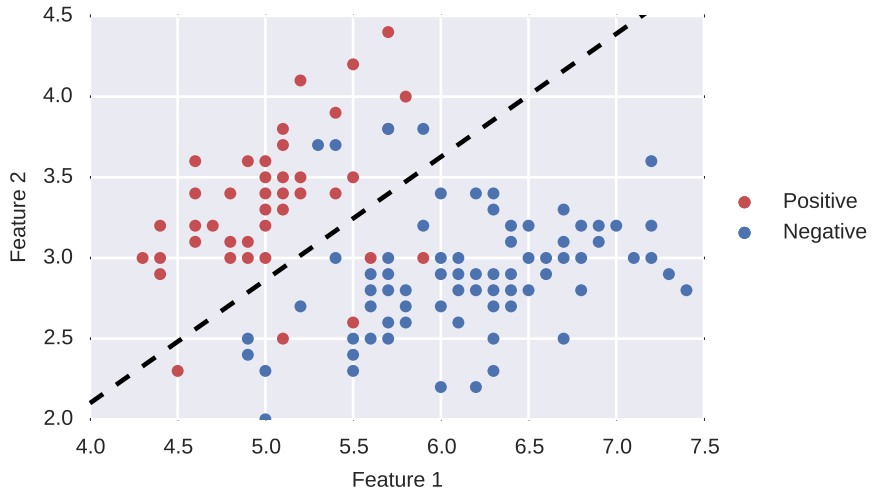
Usually, $t = \frac{1}{2}$ [Hastie et al., 2009], as an example with probability of being positive higher than 50% is classified as positive, and negative otherwise. If $t \neq \frac{1}{2}$, the function that generates the predicted class labels $\mathbf{c}$ is refers as f^t.

In Figure 2.1, the process of a training and using a classification algorithm is summarized. First, during the training phase, using a training set $\mathcal{S}_{train}$, a algorithm is train to predict $\mathbf{y}$. Then the algorithm is used to make estimate the class $\mathbf{c}$ of a set of testing examples $\mathcal{S}_{test}$.

There exists several algorithms that can be used for classification tasks. In general a classification algorithm is learn with the objective of finding patterns that separate between the different classes [Hastie et al., 2009]. In order to clarify this intuition, in Figure 2.2 an example of a classification algorithm is shown. Lets suppose a set of examples as shown in Figure 2.2a. Where the red points represents the positive examples and the blue the negative examples. The objective of a classifier is to find the best way to separate between the positive and negatives examples. In Figure 2.2b, the output of a classifier learned using



(a)



(b)

Figure 2.2: Example of a classification algorithm. Using a set of examples from two classes, a classification algorithm is learn in order to separate between the positives and the negatives.

the set of examples is shown. It is observed that this classifier is able to separate almost all the examples using a linear classifier.

However, not all examples are correctly classified. In particular there are four negative examples that were predicted as positive, and five positive examples that were predicted as negative. In the next section, we present the standard methods for evaluating the performance of a classification algorithm.

CLASSIFICATION EXAMPLES

Classification algorithms are widely used across a variety of domains. For example in the medical field, models have been used for making predictions about tumors, probability of a disease, selecting the right drug for a particular patient, and estimating the probability of relapsing, among others [Herland et al., 2014]. In the financial sector, classification models have been successfully applied for fraud detection, credit scoring, portfolio management and algorithmic trading. Also, in marketing, several models are being currently used for churn modeling, customer targeting, behavior prediction and direct marketing [Baesens, 2014]. Additionally, in many other emerging applications such as terrorism prevention, malware detection, computer security, energy consumption prediction, spam classification, and others [Kriegel et al., 2007].

2.2 PERFORMANCE EVALUATION MEASURES

When evaluating the performance of a classification algorithm, the first thing to do is to check the number of examples that were misclassified. Since the true class of the training examples is known. Therefore, evaluating the error of a model is as simply as counting the number of times an example is misclassified divided by the number of examples

$$Err(f(\mathcal{S})) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{y_i}(c_i), \tag{2.2}$$

where $\mathbf{1}_c(z)$ is an indicator function that takes the value of one if $z \in c$ and zero if $z \notin c$. Moreover the accuracy is defined as the percentage of times the algorithm made the correct prediction

$$Acc(f(\mathcal{S})) = 1 - Err(f(\mathcal{S})). \tag{2.3}$$

However, just knowing these statistics is not enough to make decisions, as in many applications is important to know where the errors are coming from. In particular, the misclassified examples may belong only to one class, which may give interesting insights about the problem. A way to observe the different errors is by looking to the confusion matrix, as shown in Table 7.1. Afterwards, using the cost matrix several statistics are extracted. In particular:

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

	Actual Positive $y = 1$	Actual Negative $y = 0$
Predicted Positive $c = 1$	True Positive (TP)	False Positive (FP)
Predicted Negative $c = 0$	False Negative (FN)	True Negative (TN)

Table 2.1: Classification confusion matrix

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.5}$$

$$F_1\text{Score} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.6}$$

As an illustrative example, the different statistics are calculated for the toy example presented in Section 2.1. First, the confusion matrix is calculated as follows:

	Actual Positive $y = 1$	Actual Negative $y = 0$
Predicted Positive $c = 1$	36	4
Predicted Negative $c = 0$	5	68

Then using the confusion matrix, the statistics are calculated,

- Error = 11.11%
- Recall = 87.8%
- Precision = 90%
- $F_1\text{Score}$ = 88.8%

There are however, several instances that are misclassified, thats because the simple linear classifier that were used in this example may not be good enough to separate between the positive and negative classes. In order to make a comparison, using the same example toy set, a new algorithm is learned. This time the algorithm made the correct prediction more often as shown in Figure 2.3. Afterwards, the confusion matrix and the different statistics are calculated as follows:

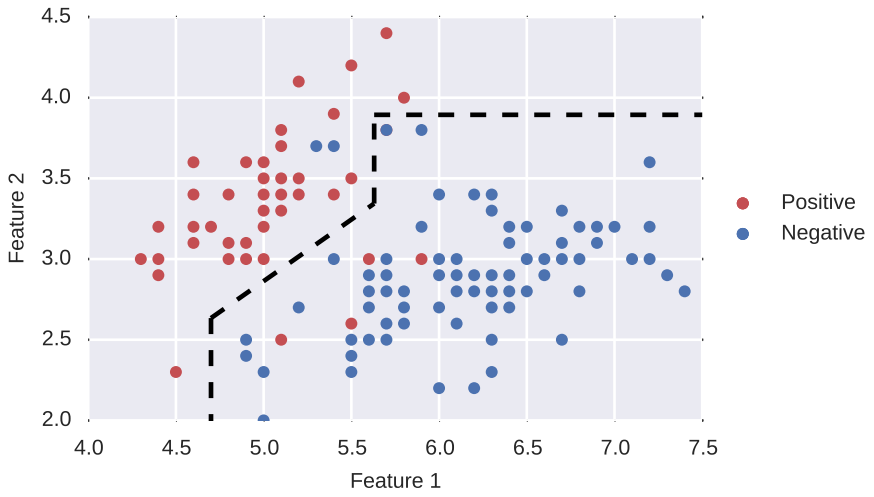


Figure 2.3: Example of a classification algorithm. Using a set of examples from two classes, a classification algorithm is learn in order to separate between the positives and the negatives.

	Actual Positive $y = 1$	Actual Negative $y = 0$
Predicted Positive $c = 1$	37	2
Predicted Negative $c = 0$	4	70

- Error = 5.3%
- Recall =90.2%
- Precision = 94.9%
- $F_1 Score$ = 92.5%

It is observed that in this case the FP are reduced more than the FN, this leads to a higher increase in precision than in recall. There is not a single rule regarding which one is more important than the other, it depends on the applications. For example in applications with a high false negative cost such as failing to identify a tumor in a medical exam, the recall should be the priority, even if that implies having a significant number of false positives. On the other hand, In applications such as spam detection, predicting a normal email as spam, may have a big impact to the customer, therefore, in this example is better to allow some false negatives and focus on the false positives.

It is not always straightforward to define the right tradeoff between false positives and false negatives. The best approximation to solve that, is to focus on the actual costs incurred by the different decisions, this is usually solved using cost-sensitive classification methods.

2.2.1 *Brier score*

Traditional evaluation measures of binary classification problems, such as Accuracy and F_1Score, provide a way to analyze the performance of a model. However, when using the classifier output as a basis for decision making, there is a need of a measure that takes into account not only the misclassification of a classifier $\mathbf{c}$, but also the quality of the estimated probability $\hat{\mathbf{p}}$ [Cohen and Goldszmidt, 2004]. The most appropriate is the Brier score [Brier, 1950]. The Brier score is one of a class of so-called proper scores which are used in evaluating the subjective probability assessment of forecasters [DeGroot and Fienberg, 1983]. The Brier score is the average squared difference between the forecasters estimated probability of and the true label:

$$BS(f(\mathcal{S})) = \frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - y_i)^2. \tag{2.7}$$

The main justification of this score, is based on decision theoretic considerations, in the sense that, a forecaster should pay a price proportional to the confidence with which it asserts its decision.

3

COST-SENSITIVE CLASSIFICATION

Outline

In this chapter, we introduce the problem of cost-sensitive classification. Standard classification models aim at minimizing the misclassification of examples, in which an example is misclassified if the predicted class is different from the true class. However, this is not the case in many real-world applications. In this chapter, first, we give an introduction to cost-sensitive classification in Section 3.1. Then, in Section 3.2, we present the problem of class-dependent cost-sensitive classification. Then, in Section 3.3, we present the general framework of example-dependent cost-sensitive classification. Within this section, we first introduce a method for defining the type of cost-sensitivity of a problem. Afterwards, we present the different cost-sensitive performance evaluation measures. Lastly, we show the state-of-the-art example-dependent cost-sensitive methods cost-proportionate rejection-sampling and cost-proportionate over-sampling.

3.1 INTRODUCTION

Classification, in the context of machine learning, deals with the problem of predicting the class of a set of examples given their features. Traditionally, classification methods aim at minimizing the misclassification of examples, in which an example is misclassified if the predicted class is different from the true class. Such a traditional framework assumes that all misclassification errors carry the same cost. This is not the case in many real-world applications. Methods that use different misclassification costs are known as cost-sensitive classifiers. Typical cost-sensitive approaches assume a constant cost for each type of error, in the sense that, the cost depends on the class and is the same among examples [Elkan, 2001; Kim et al., 2012]. Although, this class-dependent approach is not realistic in many real-world applications.

REAL-WORLD EXAMPLE-DEPENDENT COST-SENSITIVE APPLICATIONS

For example in credit card fraud detection, failing to detect a fraudulent transaction may have an economical impact from a few to thousands of Euros, depending on the particular transaction and card holder [Sahin et al., 2013]. In churn modeling, a model is used for predicting which customers are more likely to abandon a service provider. In this context, failing to identify a profitable or unprofitable churner have a significant different financial impact [Glady et al., 2009]. Similarly, in direct marketing, wrongly predicting that a customer will not accept an offer when in fact he will, has a different impact than the other way around [Zadrozny et al., 2003]. Also in credit scoring, where declining good customers has a non constant impact since not all customers generate the same profit [Verbraken et al., 2014]. Lastly, in the case of intrusion detection, classifying a benign connection as malicious have a different cost than when a malicious connection is accepted [Ma et al., 2011].

In order to deal with these specific types of cost-sensitive problems, called example-dependent cost-sensitive, some methods have been proposed recently. However, the literature on example-dependent cost-sensitive methods is limited, mostly because there is a lack of publicly available datasets that fit the problem [Aodha and Brostow, 2013]. Standard solutions consist in modifying the training set by re-weighting the examples proportionately to the misclassification costs [Elkan, 2001; Zadrozny et al., 2003].

3.2 CLASS-DEPENDENT COST-SENSITIVE CLASSIFICATION

The cost-sensitive literature is mostly focused in class-dependent problems [Elkan, 2001], where the cost of misclassification is associated with the class. Usually, the cost of misclassifying a positive example is denoted by C_{FN} and the one of misclassifying a negative example is denoted by C_{FP}. Conceptually, $C_{FN} \geqslant 0$ and $C_{FP} \geqslant 0$, moreover, it is normally defined that $C_{FN} + C_{FP} = 2$ [Flach et al., 2011], as when $C_{FN} = C_{FP} = 1$ represents the case of cost-insensitive classification. Using the previous notation, a class-dependent cost measure is defined as [Wang et al., 2014]:

$$\text{Cost}_{cd}(f(S)) = C_{FP} \cdot FP + C_{FN} \cdot FN, \tag{3.1}$$

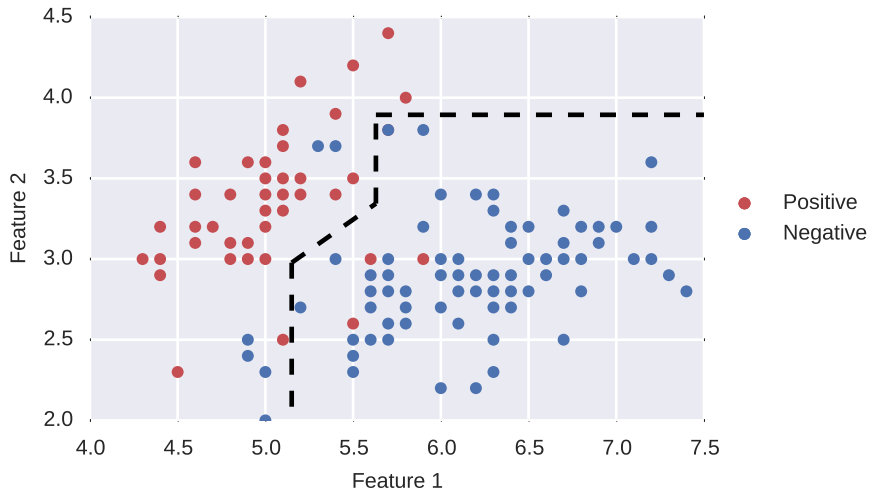


Figure 3.1: Class-dependent cost-sensitive classification algorithm. Since the cost of misclassifying positives and negatives is different, the algorithm focus on maximizing the correct classification of the positives.

where FP and FN are the number of false positives and false negatives, respectively.

Following the toy example shown in Section 2.1, we now assume that misclassifying a negative example have a cost of $C_{FN} = 0.2$ and a positive example $C_{FP} = 1.8$. Under this scenario, misclassifying a positive example have a much higher cost than misclassifying a negative one. Taking that into account, in Figure 3.1, we show an algorithm (Algorithm3), that focus on maximizing the correct classification of the positives. In the following table, we compare the results of the standard measures and the class-dependent cost, of Algorithm3 and the classification algorithms presented in Figure 2.2b (Algorithm1) and Figure 2.3 (Algorithm2):

Algorithm	Error	Recall	Precision	F_1 Score	$Cost_{cd}$
Algorithm1	11.11%	87.8%	90%	88.8%	9.8
Algorithm2	5.3%	90.2%	94.9%	92.5%	7.6
Algorithm3	7.97%	92.68%	86.36%	89.41%	6.6

It is found, that by focusing on the positives, Algorithm3 arises to a lower cost, even though, the traditional metrics are worst than Algorithm2. In conclusion, it is of highly importance to take into account the cost when evaluating and training a classification model.

Class-dependent cost-sensitive algorithms

Over the past decades, various algorithms have been proposed for class-dependent cost-sensitive classification in literature. Several authors have used modifications of decision trees that take into account the different class-dependent costs [Draper et al., 1994; Ting, 2002; Ling et al., 2004; Li et al., 2005; Kretowski and Grześ, 2006; Vadera, 2010; Lomax and Vadera, 2013]. Similarly, applications of bagging and boosting algorithms have been used for cost-sensitive classification [Nesbitt, 2010; Street, 2008; Masnadi-shirazi and Vasconcelos, 2011; Fan et al., 1999]. Recently, various variations to support vector machines have also been used for this problem [Li et al., 2010; Masnadi-shirazi, 2010]. Lastly, online learning algorithms have also been used for cost-sensitive tasks [Wang et al., 2014].

3.3 EXAMPLE-DEPENDENT COST-SENSITIVE CLASSIFICATION

The class-dependent framework introduced in the previous section, is highly restrictive, as assuming that the different costs are constant between classes is not realistic in many real world applications. In fraud detection, fraudulent transactions can have a financial impact from hundreds or thousands of Euros Sahin et al. [2013]. In this context, the example-dependent costs can be represented using a 2x2 cost matrix [Elkan, 2001], that introduces the costs associated with two types of correct classification, true positives (C_{TP_i}), true negatives (C_{TN_i}), and the two types of misclassification errors, false positives (C_{FP_i}), false negatives (C_{FN_i}), as defined in Table 3.1.

Conceptually, the cost of correct classification should always be lower than the cost of misclassification. These are referred to as the "reasonableness" conditions [Elkan, 2001], and are defined as $C_{FP_i} > C_{TN_i}$ and $C_{FN_i} > C_{TP_i}$. Taking into account the "reasonableness" conditions, a simpler cost matrix with only one degree of freedom has been defined in [Elkan, 2001], by scaling and shifting the initial cost matrix. The simpler cost-matrix is shown in Table 3.2.

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_{TP_i}	C_{FP_i}
Predicted Negative $c_i = 0$	C_{FN_i}	C_{TN_i}

Table 3.1: Classification cost matrix

Negative	$C^*_{FN_i} = \frac{(C_{FN_i} - C_{TN_i})}{(C_{FP_i} - C_{TN_i})}$
Positive	$C^*_{TP_i} = \frac{(C_{TP_i} - C_{TN_i})}{(C_{FP_i} - C_{TN_i})}$

Table 3.2: Simpler classification cost matrix [Elkan, 2001]

3.3.1 *Example-dependent evaluation measures*

Common cost-insensitive evaluation measures such as misclassification rate or F_1Score, assume the same cost for the different misclassification errors. Using these measures is not suitable for example-dependent cost-sensitive binary classification problems. Indeed, two classifiers with equal misclassification rate but different numbers of false positives and false negatives do not have the same impact on cost since $C_{FP_i} \neq C_{FN_i}$; therefore there is a need for a measure that takes into account the actual costs $\{C_{TP_i}, C_{FP_i}, C_{FN_i}, C_{TN_i}\}$ of each example i, as introduced in the previous Section.

Let S be a set of N examples i, $N = |S|$, where each example is represented by the augmented feature vector $\mathbf{x}^*_i = [\mathbf{x}_i, C_{TP_i}, C_{FP_i}, C_{FN_i}, C_{TN_i}]$ and labeled using the class label $y_i \in \{0, 1\}$. A classifier f which generates the predicted label c_i for each element i is trained using the set S. Then the cost of using f on S is calculated by

$$Cost(f(S)) = \sum_{i=1}^{N} Cost(f(\mathbf{x}^*_i)), \tag{3.2}$$

where

$$\begin{aligned} Cost(f(\mathbf{x}^*_i)) =& y_i(c_i C_{TP_i} + (1 - c_i)C_{FN_i}) + \\ & (1 - y_i)(c_i C_{FP_i} + (1 - c_i)C_{TN_i}). \end{aligned} \tag{3.3}$$

However, the total cost may not be easy to interpret. In [Whitrow et al., 2008], a *normalized* cost measure was proposed,

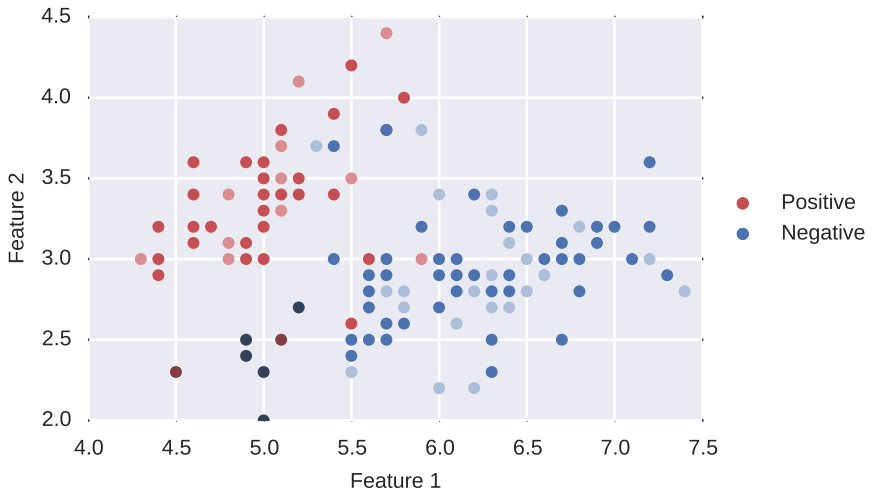


Figure 3.2: Toy example with example-dependent costs. Examples with the highest cost darker, and the ones with the lowest cost lighter.

by dividing the total cost by the theoretical maximum cost, which is the cost of misclassifying every example. The *normalized* cost is calculated using

$$\text{Cost}_n(f(\mathcal{S})) = \frac{\text{Cost}(f(\mathcal{S}))}{\sum_{i=1}^{N} C_{FN_i} \cdot \mathbf{1}_0(y_i) + C_{FP_i} \cdot \mathbf{1}_1(y_i)}. \tag{3.4}$$

We proposed similar approach in [Correa Bahnsen et al., 2014a], where the savings of using an algorithm are defined as the cost of the algorithm versus the cost of using no algorithm at all. To do that, the cost of the costless class is defined as

$$\text{Cost}_l(\mathcal{S}) = \min\{\text{Cost}(f_0(\mathcal{S})), \text{Cost}(f_1(\mathcal{S}))\}, \tag{3.5}$$

where

$$f_a(\mathcal{S}) = \mathbf{a}, \text{ with } a \in \{0, 1\}. \tag{3.6}$$

The cost improvement can be expressed as the cost savings as compared with $\text{Cost}_l(\mathcal{S})$.

$$\text{Savings}(f(\mathcal{S})) = \frac{\text{Cost}_l(\mathcal{S}) - \text{Cost}(f(\mathcal{S}))}{\text{Cost}_l(\mathcal{S})}. \tag{3.7}$$

In order to illustrate this concept, we used the same toy example shown in Section 2.1. However, now we add the variation in the costs, by showing the examples with the highest cost

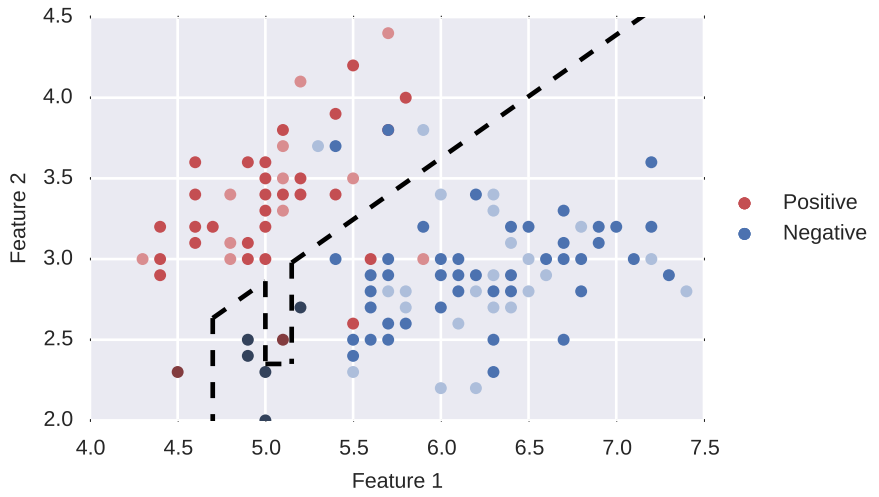


Figure 3.3: Example-dependent cost-sensitive classification algorithm. The algorithm focus first on correctly classify the dark examples, as the cost of misclassification in this cases is several times more expensive than the other cases.

darker, and the ones with the lowest cost lighter. The new example is shown in Figure 3.2. Moreover, in the following table we summarize the different example-dependent costs:

Class	Cost light	Cost normal	Cost dark
Negative	0.1	0.5	5.0
Positive	1.0	2.0	10.0

Taking into account the example-dependent costs, in Figure 3.3 (Algorithm4), we the cost when evaluating and training a classification model. show a new algorithm that focus first on correctly classify the dark examples, as the cost of misclassification in this cases is several times more expensive than the other cases. Furthermore, in the following table, we compare the results of the standard measures and the example-dependent savings, of Algorithm4 and the classification algorithms presented in Figure 2.2b (Algorithm1), Figure 2.3 (Algorithm2) and Figure 3.1 (Algorithm3):

Algorithm	Error	Recall	Precision	F_1 Score	Savings
Algorithm1	11.11%	87.8%	90%	88.8%	46.86%
Algorithm2	5.3%	90.2%	94.9%	92.5%	68.36%
Algorithm3	7.97%	92.68%	86.36%	89.41%	48.07%
Algorithm4	6.19%	92.68%	90.48%	91.56%	87.42%

With these examples, we have shown the impact that the costs have on the algorithms. Moreover, we highlight the im-

portance of using the different costs when evaluating the different models. Is worth mentioning, how distinct are the results if the costs are ignored and an algorithm is trained and evaluated not taking into account the different costs present in most real-world applications.

3.3.2 *Binary classification cost characteristic*

A classification problem is said to be cost-insensitive if costs of both errors are equal. It is class-dependent cost-sensitive if the costs are different but constant. Finally we talk about an example-dependent cost-sensitive classification problem if the cost matrix is not constant for all the examples.

However, the definition above is not general enough. There are many cases when the cost matrix is not constant and still the problem is cost-insensitive or class-dependent cost-sensitive. For example, if the costs of correct classification are zero, $C_{TP_i} = C_{TN_i} = 0$, and the costs of misclassification are $C_{FP_i} = a_0 \cdot z_i$ and $C_{FN_i} = a_1 \cdot z_i$, where a_0, a_1, are constant and z_i a random variable. This is an example of a cost matrix that is not constant. However, $C^*_{FN_i}$ and $C^*_{TP_i}$ are constant, i.e. $C^*_{FN_i} = (a_1 \cdot z_i)/(a_0 \cdot z_i) = a_1/a_0$ and $C^*_{TP_i} = 0 \; \forall i$. In this case the problem is cost-insensitive if $a_0 = a_1$, or class-dependent cost-sensitive if $a_0 \neq a_1$, even given the fact that the cost matrix is not constant.

Nevertheless, using only the simpler cost matrix is not enough to define when a problem is example-dependent cost-sensitive. To achieve this, we proposed in [Correa Bahnsen et al., 2015a], the classification problem cost characteristic as:

$$b_i = C^*_{FN_i} - C^*_{TP_i}, \tag{3.8}$$

and define its mean and standard deviation as μ_b and σ_b, respectively.

Using μ_b and σ_b, we analyze different binary classification problems. In the case of a cost-insensitive classification problem, for every example i $C_{FP_i} = C_{FN_i}$ and $C_{TP_i} = C_{TN_i}$, leading to $b_i = 1 \; \forall i$ or more generally $\mu_b = 1$ and $\sigma_b = 0$. For class-dependent cost-sensitive problems, the costs are not equal but constants $C_{FP_i} \neq C_{FN_i}$ or $C_{TP_i} \neq C_{TN_i}$, leading to $b_i \neq 1 \; \forall i$, or $\mu_b \neq 1$ and $\sigma_b = 0$. Lastly, in the case of example-dependent cost-sensitive problems, the cost difference is non constant or $\sigma_b \neq 0$.

In summary a binary classification problem is defined according to the following conditions:

μ_b	σ_b	Type of classification problem
1	0	cost-insensitive
$\neq 1$	0	class-dependent cost-sensitive
	$\neq 0$	example-dependent cost-sensitive

3.3.3 *State-of-the-art methods*

As mentioned earlier, taking into account the different costs associated with each example, some methods have been proposed to make classifiers example-dependent cost-sensitive. These methods may be grouped in two categories. Methods based on changing the class distribution of the training data, which are known as cost-proportionate sampling methods; and direct cost methods [Wang, 2013].

A standard method to introduce example-dependent costs into classification algorithms is to re-weight the training examples based on their costs, either by cost-proportionate rejection-sampling [Zadrozny et al., 2003], or over-sampling [Elkan, 2001]. The rejection-sampling approach consists in selecting a random subset $\mathcal{S}_r$ by randomly selecting examples from $\mathcal{S}$, and accepting each example i with probability $w_i / \max_{1,\dots,N}\{w_i\}$, where w_i is defined as the expected misclassification error of example i:

$$w_i = y_i \cdot C_{FN_i} + (1 - y_i) \cdot C_{FP_i}. \tag{3.9}$$

Lastly, the over-sampling method consists in creating a new set $\mathcal{S}_o$, by making w_i copies of each example i. However, cost-proportionate over-sampling increases the training since $|\mathcal{S}_o| >> |\mathcal{S}|$, and it also may result in over-fitting [Drummond and Holte, 2003]. Furthermore, none of these methods uses the full cost matrix but only the misclassification costs.

The second approach, consists in using the predicted probability $\hat{p}_i$, estimated using a given classifier f, and modify the threshold t such that the savings are maximized. This method is called cost-sensitive thresholding [Sheng and Ling, 2006]. The idea behind this approach is to adaptively modify the probability threshold of an algorithm f_t in order to maximized the

savings of the algorithm on a given set $Savings(f^t(\mathcal{S}))$. The threshold is calculated using the following equation

$$t_{thresholding} = \arg\max_t Savings(f^t(\mathcal{S})). \tag{3.10}$$

Part II

REAL-WORLD EXAMPLE-DEPENDENT APPLICATIONS

4

FINANCIAL RISK MODELING

Outline

In this chapter, we present two different real-world example-dependent cost-sensitive problems, namely, credit card fraud detection and credit scoring. Both problems are example-dependent cost-sensitive, as failing to identify a fraudulent transaction may have a financial impact ranging from tens to thousands of Euros, similarly in credit scoring, approving a customer that later do not pay his debt have a significant impact on a bank profit. First, we introduce the credit card fraud detection problem in Section 4.1. Lastly in Section 4.2, the credit scoring problem is presented.

4.1 CREDIT CARD FRAUD DETECTION

The use of credit and debit cards has increased significantly in the last years, unfortunately so has the fraud. Because of it, billions of Euros are lost every year. According to the European Central Bank [European Central Bank, 2014], during 2012 the total level of fraud reached 1.33 billion Euros in the Single Euro Payments Area, which represents an increase of 14.8% compared with 2011. Moreover, payments across non traditional channels (mobile, internet, ...) accounted for 60% of the fraud, whereas it was 46% in 2008. This opens new challenges as new fraud patterns emerge, and current fraud detection systems are not being successful in preventing fraud. Furthermore, fraudsters constantly change their strategies to avoid being detected, something that makes traditional fraud detection tools such as expert rules inadequate.

The use of machine learning in fraud detection have been an interesting topic in recent years. Several detection systems based on machine learning techniques has been successfully used for this problem [Bhattacharyya et al., 2011]. When constructing a credit card fraud detection model, there are several problems that have an important impact during the training phase: Skewness of the data , cost-sensitivity of the application, short time response of the system, dimensionality of the search

space and how to preprocess the features [Bolton et al., 2002; Bachmayer, 2008; Whitrow et al., 2008; Pozzolo et al., 2014a]. We are interested on addressing the cost-sensitivity and the features preprocessing issues.

Cost-sensitive fraud detection

Credit card fraud detection is by definition a cost-sensitive problem, in the sense that the cost due to a false positive is different than the cost of a false negative. When predicting a transaction as fraudulent, when in fact it is not a fraud, there is an administrative cost that is incurred by the financial institution. On the other hand, when failing to detect a fraud, the amount of that transaction is lost [Hand et al., 2007]. Moreover, it is not enough to assume a constant cost difference between false positives and false negatives, as the amount of the transactions vary quite significantly; therefore, its financial impact is not constant but depends on each transaction. In [Correa Bahnsen et al., 2013], we proposed a new cost-based measure to evaluate credit card fraud detection models, taking into account the different financial costs incurred by the fraud detection process.

When constructing a credit card fraud detection model, it is very important to use those features that will help the algorithm make the best decision. Typical models only use raw transactional features, such as time, amount, place of the transaction. However, this approach does not take into account the spending behavior of the customer, which is expected to help discover fraud patterns [Bachmayer, 2008]. A standard way to include these behavioral spending patters was proposed in [Whitrow et al., 2008], where Whitrow et al. proposed a transaction aggregation strategy in order to take into account a customer spending behavior. The derivation of the aggregated features consists in grouping the transactions made during the last given number of hours, first by card or account number, then by transaction type, merchant group, country or other, followed by calculating the number of transactions or the total amount spent on those transactions.

In this section, we present our previously proposed expanded version of the transaction aggregation strategy, by incorporating a combination criteria when grouping transactions, i.e., then instead of aggregating only by card holder and transaction type, we combine it with country or merchant group.

This allows to have a much richer feature space. Moreover, we are interested in analyzing the impact of adding the time of a transaction. The logic behind it, is that a customer is expected to make transactions at similar hours. We, hence, propose a new method for creating features based on the periodic behavior of a transaction time, using the von Mises distribution. In particular, these new time features should estimate if the time of a new transaction is within the confidence interval of the previous transaction time [Correa Bahnsen et al., 2015d]. Furthermore, we present the real credit card fraud dataset provided by a large European card processing company, that we used for the experiments in this thesis.

4.1.1 *Credit card fraud detection evaluation*

A credit card fraud detection algorithm consists in identifying those transactions with a high probability of being fraud, based on historical fraud patterns. The use of machine learning in fraud detection has been an interesting topic in recent years. Different detection systems that are based on machine learning techniques have been successfully used for this problem, in particular: neural networks [Maes et al., 2002], Bayesian learning [Maes et al., 2002], artificial immune systems [Bachmayer, 2008], association rules [Sánchez et al., 2009], hybrid models [Krivko, 2010], support vector machines [Bhattacharyya et al., 2011], peer group analysis [Weston et al., 2008], random forest [Correa Bahnsen et al., 2013] and online learning [Pozzolo et al., 2014a].

Most of these studies compare their proposed algorithm with a benchmark algorithm and then make the comparison using a standard binary classification measure, such as misclassification error, receiver operating characteristic (ROC), Kolmogorov-Smirnov (KS) or F_1Score statistics [Bolton et al., 2002; Hand et al., 2007; Pozzolo et al., 2014a]. However, these measures may not be the most appropriate evaluation criteria when evaluating a fraud detection models, because they tacitly assume that misclassification errors carry the same cost, similarly with the correct classified transactions. This assumption does not hold in practice, since when wrongly predicting a fraudulent transaction as legitimate carries a significantly different financial cost than the inverse case. Furthermore, the accuracy measure also assumes that the class distribution among transactions is constant and balanced [Provost et al., 1998], and typically the distri-

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	$C_{TP_i} = C_a$	$C_{FP_i} = C_a$
Predicted Negative $c_i = 0$	$C_{FN_i} = Amt_i$	$C_{TN_i} = 0$

Table 4.1: Credit card fraud cost matrix [Correa Bahnsen et al., 2013]

butions of a fraud detection data set are skewed, with a percentage of frauds ranging from 0.005% to 0.5% [Bachmayer, 2008; Bhattacharyya et al., 2011].

In order to take into account the different costs of fraud detection during the evaluation of an algorithm, we may use the cost matrix as defined in Table 3.1. Hand et al. [Hand et al., 2007] proposed a cost matrix, where in the case of false positive the associated cost is the administrative cost $C_{FP_i} = C_a$ related to analyzing the transaction and contacting the card holder. This cost is the same assigned to a true positive $C_{TP_i} = C_a$, because in this case, the card holder will have to be contacted. However, in the case of a false negative, in which a fraud is not detected, the cost is defined to be a hundred times larger, i.e. $C_{FN_i} = 100C_a$. This same approach was also used in [Bachmayer, 2008].

Nevertheless, in practice, losses due to a specific fraud range from few to thousands of Euros, which means that assuming constant cost for false negatives is unrealistic. In order to address this limitation, in [Correa Bahnsen et al., 2013], we proposed a cost matrix that takes into account the actual example-dependent financial costs. Our cost matrix defines the cost of a false negative to be the amount $C_{FN_i} = Amt_i$ of the transaction i. We argue that this cost matrix is a better representation of the actual costs, since when a fraud is not detected, the losses of that particular fraud correspond to the stolen amount. The costs are summarized in Table 4.1.

Afterwards, using (3.3) a cost measure for fraud detection is calculated as:

$$Cost(f(S)) = \sum_{i=1}^{N} y_i(1 - c_i)Amt_i + c_i C_a \tag{4.1}$$

then, the savings of an algorithm are calculated using (3.7).

Attribute name	Description
Transaction ID	Transaction identification number
Time	Date and time of the transaction
Account number	Identification number of the customer
Card number	Identification of the credit card
Transaction type	ie. Internet, ATM, POS, ...
Entry mode	ie. Chip and pin, magnetic stripe, ...
Amount	Amount of the transaction in Euros
Merchant code	Identification of the merchant type
Merchant group	Merchant group identification
Country	Country of trx
Country 2	Country of residence
Type of card	ie. Visa debit, Mastercard, American Express...
Gender	Gender of the card holder
Age	Card holder age
Bank	Issuer bank of the card
Fraud	Indicator whenever or not a transaction was a fraud

Table 4.2: Summary of typical raw credit card fraud detection features

4.1.2 *Feature engineering for fraud detection*

When constructing a credit card fraud detection algorithm, the initial set of features (raw features) include information regarding individual transactions. It is observed throughout the literature, that regardless of the study, the set of raw features is quite similar. This is because the data collected during a credit card transaction must comply the international financial reporting standards [American Institute of CPAs, 2011]. In Table 4.2, the typical credit card fraud detection raw features are summarized.

4.1.2.1 *Customer spending patterns*

Several studies use only the raw features in carrying their analysis [Brause et al., 1999; Minegishi and Niimi, 2011; Panigrahi et al., 2009; Sánchez et al., 2009]. However, as noted in [Bolton and Hand, 2001], a single transaction information is not sufficient to detect a fraudulent transaction, since as using only the raw features leaves behind important information such as the consumer spending behavior, which is usually used by commercial fraud detection systems [Whitrow et al., 2008].

To deal with this, in [Bachmayer, 2008], a new set of features were proposed such that the information of the last transaction made with the same credit card is also used to make a prediction. The objective, is to be able to detect very dissimilar continuous transactions within the purchases of a customer. The new set of features include: time since the last transaction, previous amount of the transaction, previous country of the transaction. Nevertheless, these features does not take into account consumer behavior other than the last transaction made by a client, this leads to having an incomplete profile of customers.

A more compressive way to take into account a customer spending behavior is to derive some features using a transaction aggregation strategy. This methodology was initially proposed in [Whitrow et al., 2008]. The derivation of the aggregation features consists in grouping the transactions made during the last given number of hours, first by card or account number, then by transaction type, merchant group, country or other, followed by calculating the number of transactions or the total amount spent on those transactions. This methodology has been used by a number of studies [Bhattacharyya et al., 2011; Weston et al., 2008; Tasoulis and Adams, 2008; Correa Bahnsen et al., 2013; Sahin et al., 2013; Correa Bahnsen et al., 2014b; Pozzolo et al., 2014a].

When aggregating a customer transactions, there is an important question on how much to accumulate, in the sense that the marginal value of new information may diminish as time passes. Whitrow et al. [2008], discuss that aggregating 101 transactions is not likely to be more informative than aggregating 100 transactions. Indeed, when time passes, information lose their value, in the sense that a customer spending patterns are not expected to remain constant over the years. In particular, Whitrow et al. define a fixed time frame to be 24, 60 or 168 hours.

The process of aggregating features consists in selecting those transactions that were made in the previous t_p hours, for each transaction i in the dataset $\mathcal{S}$,

$$\mathcal{S}_{agg} \equiv \text{TRX}_{agg}(\mathcal{S}, i, t_p) = \left\{ x_l^{amt} \middle| \left(x_l^{id} = x_i^{id} \right) \wedge \left(\text{hours}(x_i^{time}, x_l^{time}) < t_p \right) \right\}_{l=1}^{N}, \quad (4.2)$$

where $N = |\mathcal{S}|$, $|\cdot|$ being the cardinality of a set, x_i^{time} is the time of transaction i, x_i^{amt} is the amount of transaction i, x_i^{id} the cus-

tomer identification number of transaction i, and $\mathrm{hours}(t_1, t_2)$ is a function that calculates the number of hours between the times t_1 and t_2. Afterwards the feature number of transactions and amount of transactions in the last t_p hours are calculated as:

$$x_i^{a1} = |\mathcal{S}_{agg}|, \tag{4.3}$$

and

$$x_i^{a2} = \sum_{x^{amt} \in \mathcal{S}_{agg}} x^{amt}, \tag{4.4}$$

respectively.

We note that this aggregation is not enough, in the sense that the combination of different features is not being taken into account. For example, it is not only interesting to see the total transactions, but also group them following a certain criteria, such as: transactions made in the last t_p hours, in the same country and of the same transaction type. For calculating such features, first we expand (4.2) as follows

$$\begin{aligned} \mathcal{S}_{agg2} \equiv & \mathrm{TRX}_{agg}(\mathcal{S}, i, t_p, cond_1, cond_2) = \Big\{ x_l^{amt} \Big| \\ & \left(x_l^{id} = x_i^{id}\right) \wedge \left(\mathrm{hours}(x_i^{time}, x_l^{time}) < t_p\right) \wedge \\ & \left(x_l^{cond_1} = x_i^{cond_1}\right) \wedge \left(x_l^{cond_2} = x_i^{cond_2}\right) \Big\}_{l=1}^{N}. \end{aligned} \tag{4.5}$$

where, $cond_1$ and $cond_2$, could be either of the features of a transaction listed in Table 4.2. Then, the features are calculated as:

$$x_i^{a3} = |\mathcal{S}_{agg2}|, \tag{4.6}$$

and

$$x_i^{a4} = \sum_{x^{amt} \in \mathcal{S}_{agg2}} x^{amt}. \tag{4.7}$$

To further clarify how the aggregated features are calculated we show an example. Consider a set of transactions made by a client between the first and third of January of 2015, as shown in Table 4.3. Then we estimate the aggregated features (x_i^{a1}, x_i^{a2}, x_i^{a3} and x_i^{a4}) by setting $t_p = 24$ hours. The different aggregated features give us different information of the customer

Raw features					Aggregated features			
Id	**Time**	**Type**	**Country**	**Amt**	x_i^{a1}	x_i^{a2}	x_i^{a3}	x_i^{a4}
1	01/01 18:20	POS	LUX	250	0	0	0	0
2	01/01 20:45	POS	LUX	400	1	250	1	250
3	01/01 22:40	ATM	LUX	250	2	650	0	0
4	02/01 00:50	POS	GER	50	3	900	0	0
5	02/01 19:18	POS	GER	100	3	700	1	50
6	02/01 23:45	POS	GER	150	2	150	2	150
7	03/01 06:00	POS	LUX	10	3	400	0	0

Table 4.3: Example calculation of aggregated features. For all aggregated features $t_p = 24$.

spending behavior. Moreover, the total number of aggregated features can grow quite quickly, as t_p can have several values, and the combination of combination criteria can be quite large as well. In Correa Bahnsen et al. [2013], we used a total of 280 aggregated features. In particular we set the different values of t_p to: 1, 3, 6, 12, 18, 24, 72 and 168 hours. Then calculate the aggregated features using (4.2), and also using (4.5) with the following grouping criteria: country, type of transaction, entry mode, merchant code and merchant group.

4.1.2.2 *Time features*

When using the aggregated features, there is still some information that is not completely captured by those features. In particular we are interesting in analyzing the time of the transaction. The logic behind this, is that a customer is expected to make transactions at similar hours. The issue when dealing with the time of the transaction, specifically, when analyzing a feature such as the mean of transactions time, is that it is easy to make the mistake of using the arithmetic mean. Indeed the arithmetic mean is not a correct way to average time, because as shown in Figure 4.1, it does not take into account the periodic behavior of the time feature. For example, the arithmetic mean of transaction time of four transactions made at 2:00, 3:00, 22:00 and 23:00 is 12:30, which is counter intuitive since no transaction was made close to that time.

We propose to overcome this limitation by modeling the time of the transaction as a periodic variable, in particular using the von Mises distribution. The von Mises distribution, also known as the periodic normal distribution, is a distribu-

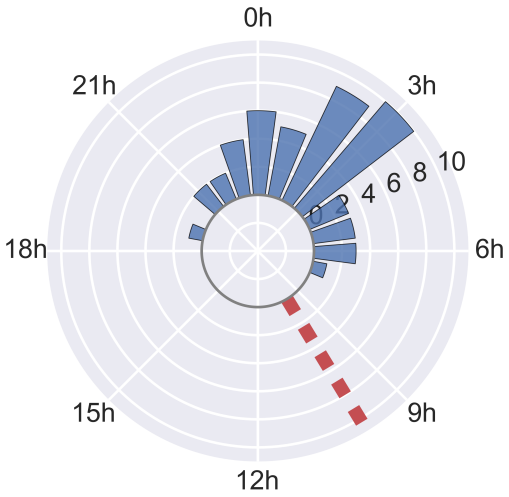


Figure 4.1: Analysis of the time of a transaction using a 24 hour clock. The arithmetic mean of the transactions time (dashed line) do not accurately represents the actual times distribution.

tion of a wrapped normal distributed variable across a circle [Fisher, 1996]. The von Mises distribution of a set of examples $D = \{t_1, t_2, \cdots, t_N\}$ is defined as

$$D \sim vonmises\left(\mu_{vM}, \frac{1}{\sigma_{vM}}\right), \tag{4.8}$$

where μ_{vM} and σ_{vM} are the periodic mean and periodic standard deviation, respectively, and are defined as

$$\mu_{vM}(D) = 2 \cdot \arctan\left(\frac{\phi}{\left(\sqrt{\psi^2 + \phi^2} + \psi\right)}\right), \tag{4.9}$$

and

$$\sigma_{vM}(D) = \sqrt{\ln\left(\frac{1}{\left(\frac{\phi}{N}\right)^2 + \left(\frac{\psi}{N}\right)^2}\right)}, \tag{4.10}$$

respectively. Where $\phi = \sum_{t_j \in D} \sin(t_j)$ and $\psi = \sum_{t_j \in D} \cos(t_j)$ [Bishop, 2006].

In particular we are interested in calculating a confidence interval (CI) for the time of a transaction. For doing that, initially we select a set of transactions made by the same client in the last t_p hours,

$$\mathcal{S}_{per} \equiv TRX_{vM}(\mathcal{S}, i, t_p) = \left\{ x_l^{time} \middle| \left(x_l^{id} = x_i^{id}\right) \wedge \right.$$

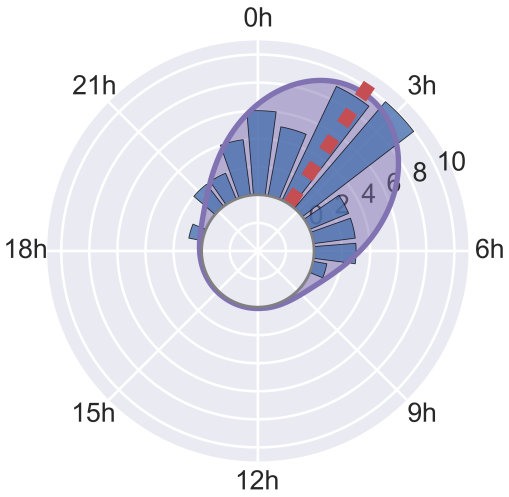


Figure 4.2: Fitted von Mises distribution including the periodic mean (dashed line) and the probability distribution (purple area).

$$\left(\text{hours}(x_i^{\text{time}}, x_l^{\text{time}}) < t_p\right) \Big\}_{l=1}^{N}. \tag{4.11}$$

Afterwards, the probability distribution function of the time of the set of transactions is calculated as:

$$x_i^{\text{time}} \sim \text{vonmises}\left(\mu_{vM}(\mathcal{S}_{\text{per}}), \frac{1}{\sigma_{vM}(\mathcal{S}_{\text{per}})}\right). \tag{4.12}$$

In Figure 4.2, the von Mises distribution calculation is shown. It is observed that the arithmetic mean is quite different from the periodic mean, the latter being a more realistic representation of the actual transaction times. Then, using the estimated distribution a new set of features can be extracted, ie. a binary feature (x_i^{p1}) if a new transaction time is within the confidence interval range with probability α. An example is presented in Figure 4.3. Furthermore, other features can be calculated, as the confidence interval range can be calculated for several values of α, and also the time period can have an arbitrary size.

Additionally, following the same example presented on Table 4.3, we calculate a feature x_i^{p1}, as a binary feature that takes the value of one if the current time of the transaction is within the confidence interval of the time of the previous transactions with a confidence of $\alpha = 0.9$. The example is show in Table 4.4. In this example, it is shown, how the arithmetic and periodic

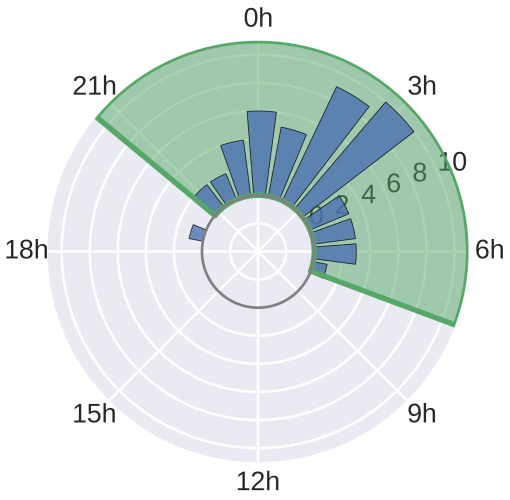


Figure 4.3: Expected time of a transaction (green area). Using the confidence interval, a transaction can be flag normal or suspicious, depending whether or not the time of the transaction is within the confidence interval.

Raw features		Arithmetic	Periodic features		
Id	**Time**	**mean**	**mean**	**Confidence interval**	x_i^{p1}
1	01/01 18:20	—	—	—	—
2	01/01 20:45	—	—	—	—
3	01/01 22:40	19:27	19:27	15:45 - 23:10	True
4	02/01 00:50	20:28	20:28	17:54 - 23:03	False
5	02/01 19:18	16:44	22:44	18:51 - 00:17	True
6	02/01 23:45	16:19	21:07	15:21 - 02:52	True
7	03/01 06:00	18:43	22:43	17:19 - 01:46	False

Table 4.4: Example calculation of periodic features.

means differ, as for the last transaction both means are significantly different. Moreover, the new feature, help to get a felling of when a customer is expected to make transactions.

Finally, when calculating the periodic features, it is important to use longer time frames t_p, since if the distribution is calculated using only a couple of transactions it may not be as relevant of a customer behavior patterns, compared against using a full year of transactions. Evidently, if t_c is less than 24 hours, any transaction made afterwards will not be expected to be within the distribution of previous transactions times. To avoid this we recommend using at least the previous 7 days of transactional information, therefore, having a better understanding of its behavioral patterns. Lastly, this approach can also be used

to estimate features such as the expected day of the week of transactions, as some customers may only use their credit cards during the weekend nights, or during working hours.

4.1.3 *Database*

For this thesis we used a dataset provided by a large European card processing company. The dataset consists of fraudulent and legitimate transactions made with credit and debit cards between January 2012 and June 2013. The total dataset contains 120,000,000 individual transactions, each one with 27 attributes, including a fraud label indicating whenever a transaction is identified as fraud. This label was created internally in the card processing company, and can be regarded as highly accurate. In the dataset only 40,000 transactions were labeled as fraud, leading to a fraud ratio of 0.025%.

4.2 CREDIT SCORING

In order to mitigate the impact of credit risk and make more objective and accurate decisions, financial institutions use credit scores to predict and control their losses. The objective in credit scoring is to classify which potential customers are likely to default a contracted financial obligation based on the customer's past financial experience, and with that information decide whether to approve or decline a loan [Anderson, 2007]. This tool has become a standard practice among financial institutions around the world in order to predict and control their loans portfolios. When constructing credit scores, it is a common practice to use standard cost-insensitive binary classification algorithms such as logistic regression, neural networks, discriminant analysis, genetic programing, decision trees, among others [Hand and Henley, 1997; Correa Bahnsen and Gonzalez Montoya, 2011].

Formally, a credit score is a statistical model that allows the estimation of the probability $\hat{p}_i = P(y_i = 1|\mathbf{x}_i)$ of a customer i defaulting a contracted debt. Additionally, since the objective of credit scoring is to estimate a classifier c_i to decide whether or not to grant a loan to a customer i, a threshold t is defined such that if $\hat{p}_i < t$, then the loan is granted, i.e., $c_i(t) = 0$, and denied otherwise, i.e., $c_i(t) = 1$.

There exists different approaches for defining the probability threshold. The sensitivity versus specificity (SvsS) approach

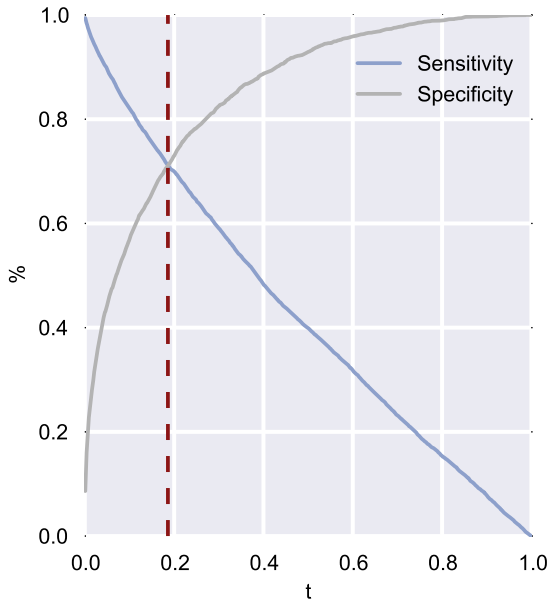


Figure 4.4: Credit scoring sensitivity versus specificity thresholding procedure.

is the most widely used among financial institutions [Anderson, 2007], where specificity is the true positive rate $F_0(t)$ for a threshold t, and the sensitivity is one minus the false positive rate $F_1(t)$ given a threshold t [Hernandez-Orallo et al., 2012]. In this method the objective is to fix the threshold at the point where the sensitivity is equal to the specificity $F_0(t) = 1 - F_1(t)$, where $F_0(t)$ and $F_1(t)$ are calculated using:

$$F_a(t) = \frac{1}{N_a} |\{\mathbf{x}_i | \mathbf{x}_i \in \mathcal{S}_a \wedge \hat{p}_i \leqslant t\}|, \text{ for } a \in \{0, 1\}. \tag{4.13}$$

Lastly, the *SvsS* threshold t_{SvsS} is found by using

$$t_{SvsS} = \arg\min_t |F_0(t) - (1 - F_1(t))|. \tag{4.14}$$

This process is further clarify in Figure 4.4.

After the classifier c_i is estimated, there is a need to evaluate its performance. In practice, many statistical evaluation measures are used to assess the performance of a credit scoring model. Measures such as the area under the receiver operating characteristic curve (AUC), Brier score, Kolmogorov-Smirnoff (K-S) statistic, F_1-Score, and misclassification are among the most common [Beling et al., 2005]. Nevertheless, none of these measures takes into account the business and economical realities that take place in credit scoring. Costs that the financial

institution had incurred to acquire customers, or the expected profit due to a particular client, are not considered in the evaluation of the different models. This is explored in detail in the next section.

4.2.1 *Financial evaluation of a credit scorecard*

Typically, a credit risk model is evaluated using standard cost-insensitive measures. However, in practice, the cost associated with approving what is known as a bad customer, i.e., a customer who default his credit loan, is quite different from the cost associated with declining a good customer, i.e., a customer who successfully repay his credit loan. Furthermore, the costs are not constant among customers. This is because loans have different credit line amounts, terms, and even interest rates. Some authors have proposed methods that include the misclassification costs in the credit scoring context [Verbraken et al., 2014; Alejo and Garc, 2013; Beling et al., 2005; Oliver and Thomas, 2009]. However, they assume a constant misclassification cost, which is not the case in credit scoring.

Initial approaches to include the different costs have been published in recent years, particularly the one proposed by Beling et al. [Beling et al., 2005; Oliver and Thomas, 2009], in which the costs of misclassification are assigned for each error. Specifically, setting the cost of a false positive C_{FP} to the loan's annual interest rate charged to the customer int_r, the cost of a false negative C_{FN} to the loss given default L_{gd}, which is the percentage of loss over the total credit line when the customer defaulted, and setting to zero the costs of true positive C_{TP} and true negative C_{TN}. Using that, they proposed the expected cost (EC) method to find the probability threshold that minimizes those costs, $t_{ec} = \frac{C_{FN}}{C_{FN}+C_{FP}} = \frac{L_{gd}}{L_{gd}+int_r}$. Nevertheless, this approach assumes a constant cost within examples, which is a strong assumption, since in practice each example carries a very different cost, given by the different credit limits and conditions of each loan. Consequently, there is a need for an example-dependent cost matrix that takes into account the cost of misclassifying each example.

In order to take into account the varying costs that each example carries, we proposed in [Correa Bahnsen et al., 2014a], a cost matrix with example-dependent misclassification costs as given in Table 4.5. First, we assume that the costs of a correct classification, C_{TP_i} and C_{TN_i}, are zero for every customer

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	$C_{TP_i} = 0$	$C_{FP_i} = r_i + C^a_{FP}$
Predicted Negative $c_i = 0$	$C_{FN_i} = Cl_i \cdot L_{gd}$	$C_{TN_i} = 0$

Table 4.5: Credit scoring example-dependent cost matrix

i. We define C_{FN_i} to be the losses if the customer i defaults to be proportional to his credit line Cl_i. We define the cost of a false positive per customer C_{FP_i} as the sum of two real financial costs r_i and C^a_{FP}, where r_i is the loss in profit by rejecting what would have been a good customer.

The profit per customer r_i is calculated as the present value of the difference between the financial institution gains and expenses, given the credit line Cl_i, the term l_i and the financial institution lending rate int_{r_i} for customer i, and the financial institution of cost funds int_{cf}.

$$r_i = PV(A(Cl_i, int_{r_i}, l_i), int_{cf}, l_i) - Cl_i, \tag{4.15}$$

with A being the customer monthly payment and PV the present value of the monthly payments, which are calculated using the time value of money equations [Lawrence and Solomon, 2012],

$$A(Cl_i, int_{r_i}, l_i) = Cl_i \frac{int_{r_i}(1 + int_{r_i})^{l_i}}{(1 + int_{r_i})^{l_i} - 1}, \tag{4.16}$$

$$PV(A, int_{cf}, l_i) = \frac{A}{int_{cf}} \left(1 - \frac{1}{(1 + int_{cf})^{l_i}}\right). \tag{4.17}$$

The second term C^a_{FP}, is related to the assumption that the financial institution will not keep the money of the declined customer idle. It will instead give a loan to an alternative customer [Nayak and Turvey, 1997]. Since no further information is known about the alternative customer, it is assumed to have an average credit line $\overline{Cl}$ and an average profit $\bar{r}$. Given that,

$$C^a_{FP} = -\bar{r} \cdot \pi_0 + \overline{Cl} \cdot L_{gd} \cdot \pi_1, \tag{4.18}$$

in other words minus the profit of an average alternative customer plus the expected loss, taking into account that the alternative customer will pay his debt with a probability equal to the prior negative rate, and similarly will default with probability equal to the prior positive rate.

4.2.1.1 *Calculation of the credit limit*

One key parameter of our model is the credit limit. There exists several strategies to calculate the Cl_i depending on the type of loans, the state of the economy, the current portfolio, among others [Anderson, 2007; Lawrence and Solomon, 2012]. Nevertheless, given the lack of information regarding the specific business environments of the considered datasets, we simply define Cl_i as

$$Cl_i = \min\left\{q \cdot Inc_i, Cl_{max}, Cl_{max}(debt_i)\right\}, \tag{4.19}$$

where Inc_i and $debt_i$ are the monthly income and debt ratio of the customer i, respectively, q is a parameter that defines the maximum Cl_i in times Inc_i, and Cl_{max} the maximum overall credit line. Lastly, the maximum credit line given the current debt is calculated as the maximum credit limit such that the current debt ratio plus the new monthly payment does not surpass the customer monthly income. It is calculated as

$$Cl_{max}(debt_i) = PV\left(Inc_i \cdot P_m(debt_i), int_{r_i}, l_i\right), \tag{4.20}$$

and

$$P_m(debt_i) = \min\left\{\frac{A(q \cdot Inc_i, int_{r_i}, l_i)}{Inc_i}, (1 - debt_i)\right\}. \tag{4.21}$$

4.2.2 *Databases*

For this thesis we used two different publicly available credit scoring datasets. The first dataset is the **2011 Kaggle competition Give Me Some Credit**[1], in which the objective is to identify those customers of personal loans that will experience financial distress in the next two years. The second dataset is from the **2009 Pacific-Asia Knowledge Discovery and Data Mining conference (PAKDD) competition**[2]. Similarly, this competition had the objective of identifying which credit card applicants were likely to default and by doing so deciding whether or not to approve their applications. The Kaggle Credit and PAKDD Credit datasets contain information regarding the features, and more importantly about the income of each example, from which an estimated credit limit Cl_i can be calculated.

1 http://www.kaggle.com/c/GiveMeSomeCredit/
2 http://sede.neurotech.com.br:443/PAKDD2009/

Parameter	Kaggle Credit	PAKDD Credit
Interest rate (int_r)	4.79%	63.0%
Cost of funds (int_{cf})	2.94%	16.5%
Term (l) in months	24	24
Loss given default (L_{gd})	75%	75%
Times income (q)	3	3
Maximum credit line (Cl_{max})	25,000	25,000

Table 4.6: Credit scoring model parameters

The Kaggle Credit dataset contains 112,915 examples, each one with 10 features and the class label. The proportion of default or positive examples is 6.74%. On the other hand, the PAKDD Credit dataset contains 38,969 examples, with 30 features and the class label, with a proportion of 19.88% positives. This database comes from a Brazilian financial institution, and as it can be inferred from the competition description, the data was obtained around 2004.

Since no specific information regarding the datasets is provided, we assume that they belong to average European and Brazilian financial institutions. This enabled us to find the different parameters needed to calculate the cost measure. In Table 4.6, the different parameters are shown. In particular, we obtain the average interest rates in Europe during 2013 from the European Central Bank [ECB, 2014], and the average interest and exchange rates in Brazil during 2004 from Trading Economics [Economics, 2014]. Because the income is not in the same currency on both datasets, we convert the PAKDD Credit dataset to Euros. Additionally, we use a fixed loan term l for both datasets, considering that in the Kaggle Credit dataset the class was constructed to predict two years of credit behavior, and because the PAKDD Credit dataset is related to credit cards the term is fix to two years [Lawrence and Solomon, 2012]. Moreover, we set the loss given default L_{gd} using information from the Basel II standard[3], q to 3 since it is the average personal loan requests related to monthly income, and the maximum credit limit Cl_{max} to 25,000 Euros.

3 http://www.bis.org/publ/bcbsca.htm.

5

MARKETING ANALYTICS

Outline

In this chapter, we present two different marketing real-world example-dependent cost-sensitive problems, namely, churn modeling and direct marketing. Both problems deal with identifying those customers with certain characteristic with the objective to maximize the results of the different CRM strategies. First, we introduce the churn modeling problem in Section 5.1. Lastly in Section 5.2, the direct marketing problem is presented.

5.1 CHURN MODELING

Customer churn predictive modeling deals with predicting the probability of a customer defecting using historical, behavioral and socio-economical information. This tool is of great benefit to subscription based companies allowing them to maximize the results of retention campaigns. The problem of churn predictive modeling has been widely studied by the data mining and machine learning communities. It is usually tackled by using classification algorithms in order to learn the different patterns of both the churners and non-churners. Nevertheless, current state-of-the-art classification algorithms are not well aligned with commercial goals, in the sense that, the models miss to include the real financial costs and benefits during the training and evaluation phases. In the case of churn, evaluating a model based on a traditional measure such as accuracy or predictive power, does not yield to the best results when measured by the actual financial cost, ie. investment per subscriber on a loyalty campaign and the financial impact of failing to detect a real churner versus wrongly predicting a non-churner as a churner.

In this section, we present a cost-sensitive framework for customer churn predictive modeling, that we proposed in [Correa Bahnsen et al., 2015b]. First, in Section 5.1.1, we introduce the problem of chrun modeling. Then in Section 5.1.2, we present a financial based measure for evaluating the effectiveness of a churn campaign taking into account the available portfolio of

offers, their individual financial cost and probability of offer acceptance depending on the customer profile. Finally, in Section 5.1.3, we describe the real-world churn modeling dataset that will be used during the experiments.

5.1.1 *Introduction*

The two main objectives of subscription-based companies are to acquire new subscribers and retain those they already have, mainly because profits are directly linked with the number of subscribers. In order to maximize the profit, companies must increase the customer base by incrementing sales while decreasing the number of churners. Furthermore, it is common knowledge that retaining a customer is about five times less expensive than acquiring a new one [Farris et al., 2010], this creates pressure to have better and more effective churn campaigns.

A typical churn campaign consists in identifying from the current customer base which ones are more likely to leave the company, and make an offer in order to avoid that behavior. With this in mind the companies use intelligence to create and improve retention and collection strategies. In the first case, this usually implies an offer that can be either a discount or a free upgrade during certain span of time. In both cases the company has to assume a cost for that offer, therefore, accurate prediction of the churners becomes important. The logic of this flow is shown in Figure 5.1.

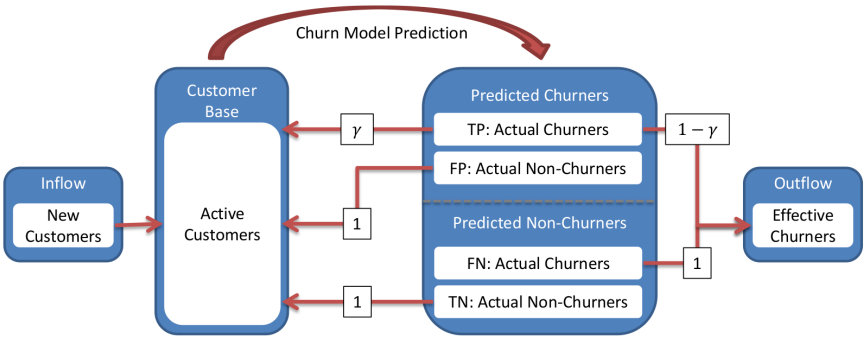


Figure 5.1: Flow analysis of a churn campaign [Verbraken, 2012]

Flow analysis of a churn campaign

The typical churn campaign process starts with the sales that every month increase the customer base, however, monthly there is a group of customers that decide to leave the company for many reasons. Then the objective of a churn model is to identify those customers before they take the decision of defecting.

Using a churn model, those customers more likely to leave are predicted as churners and an offer is made in order to retain them. However, it is known that not all customers will accept the offer, in the case when a customer is planning to defect, it is possible that the offer is not good enough to retain him or that the reason for defecting can not be influenced by an offer. Using historical information, it is estimated that a customer will accept the offer with probability γ. On the other hand, there is the case in which the churn model misclassified a non-churner as churner, also known as false positives, in that case the customer will always accept the offer that means and additional cost to the company since those misclassified customers do not have the intentions of leaving.

In the case were the churn model predicts customers as non-churners, there is also the possibility of a misclassification, in this case an actual churner is predicted as non-churner, since these customers do not receive an offer and they will leave the company, these cases are known as false negatives. Lastly, there is the case were the customers are actually non-churners, then there is no need to make a retention offer to these customers since they will continue to be part of the customer base.

It can be seen that a churn campaign (or churn model) have three main points. First, avoid false positives since there is a financial cost of making an offer were it is not needed. Second, to the true positives, give the right offer that maximize γ while maximizing the profit of the company. And lastly, to decrease the number of false negatives.

From a machine learning perspective, a churn model is a classification algorithm. In the sense that using historical information, a prediction of which current customers are more likely to defect, is made. This model is normally created using one of a number of well establish algorithms (Logistic regression, neural networks, random forests, among others) [Ngai et al., 2009; KhakAbi et al., 2010]. Then, the model is evaluated using mea-

sures such as misclassification error, receiver operating characteristic (ROC), Kolmogorov-Smirnov (KS) or F_1Score statistics [Verbeke et al., 2012]. However these measures assume that misclassification errors carry the same cost, which is not the case in churn modeling, since failing to identify a profitable or unprofitable churner have significant different financial costs [Glady et al., 2009].

In the next section, we propose a new financial based measure for evaluating the effectiveness of a voluntary churn campaign taking into account the available portfolio of offers, their individual financial cost and probability of acceptance depending on the customer profile.

5.1.2 *Evaluation of a churn campaign*

Traditionally, a churn model is evaluated as a standard binary classification model, using measures such as misclassification error, receiver operating characteristic (ROC), Kolmogorov-Smirnov (KS) or F_1Score statistics [Verbeke et al., 2012]. However, these measures may not be the most appropriate evaluation criteria when evaluating a churn model, because they tacitly assume that misclassification errors carry the same cost, similarly with the correct classified examples. This assumption does not hold in many real-world applications such as churn modeling, since when misidentifying a churner the financial losses are quite different than when misclassifying a non-churner as churner [Glady et al., 2009]. Furthermore, the accuracy measure also assumes that the class distribution among examples is constant and balanced [Provost et al., 1998], and typically the distributions of a churn data set are skewed [Verbeke et al., 2012].

Different studies have proposed measures to deal with these cost-sensitivity related to evaluating a churn model. In [Neslin et al., 2006], a profit-based measure was proposed by starting with the confusion matrix and multiplying it with the expected profit of each case.

$$\begin{aligned}\text{Profit}_1 = (\text{TP}+\text{FP})\Big[&(\gamma \text{CLV} + C_o(1-\gamma)(-C_a))\,\pi_1\gamma \\ &- C_o - C_a\Big] - N \cdot C_a, \end{aligned} \tag{5.1}$$

with $N \cdot C_a$ being the fixed administrative cost of running the campaign, C_o the average cost of the retention offer, C_a the cost of contacting the customer, π_1 the prior churn rate and CLV the average customer lifetime value. Moreover, as discussed in [Verbraken et al., 2013], if the average instead of the total profit is considered and the fixed cost $N \cdot C_a$ is discarded since is irrelevant for classifier selection, the profit can be expressed as:

$$\begin{aligned}\text{Profit}_2 =&\text{TP}\,(\gamma(\text{CLV} - C_o - C_a) + (1-\gamma)C_a) \\ &+ \text{FP}(-C_o - C_a). \end{aligned} \tag{5.2}$$

Nevertheless, equations (5.1) and (5.2), assume that every customer has the same CLV and C_o, whereas this is not true in practice. In fact, different customers have a very different CLV, and not all offers can be made to every customer, neither do they have the same impact across customers. In order to obtain a more business oriented measure, we first analyze the financial impact of the different decisions, ie. false positives, false negatives, true positives and true negatives, for each customer. In Figure 5.2, the financial impact of a churn model is shown. Note than we take into account the costs and not the profit in each case.

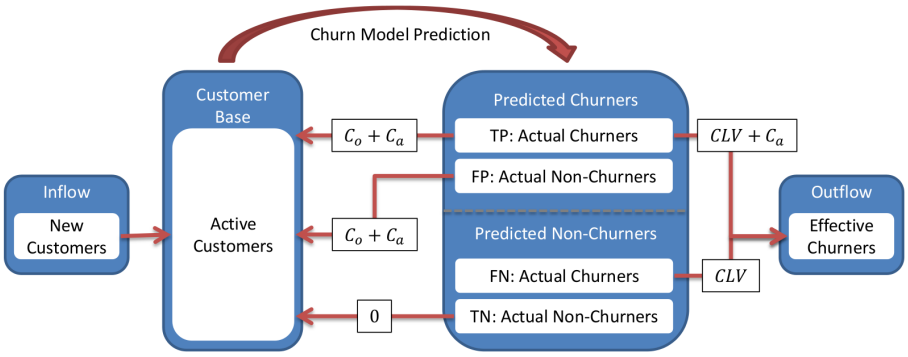


Figure 5.2: Financial impact of the different decisions ie. False positives, false negatives, true positives and true negatives

Financial analysis of a churn champaign

When a customer is predicted to be a churner, an offer is made with the objective of avoiding the customer defecting. However, if a customer is actually a churner, he may or not accept the offer with a probability γ_i. If the customer accepts the offer, the financial impact is equal to the cost of the offer (C_{o_i}) plus the administrative cost of contacting the customer (C_a). On the other hand, if the customer declines the offer, the cost is the expected income that the clients would otherwise generate, also called customer lifetime value (CLV_i), plus C_a. Lastly, if the customer is not actually a churner, he will be happy to accept the offer and the cost will be C_{o_i} plus C_a.

In the case that the customer is predicted as non-churner, there are two possible outcomes. Either the customer is not a churner, then the cost is zero, or the customer is a churner and the cost is CLV_i.

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	$C_{TP_i} = \gamma_i C_{o_i} + (1-\gamma_i)(CLV_i + C_a)$	$C_{FP_i} = C_{o_i} + C_a$
Predicted Negative $c_i = 0$	$C_{FN_i} = CLV_i$	$C_{TN_i} = 0$

Table 5.1: Proposed churn modeling example-dependent cost matrix

The different costs are summarized in Table 5.1. Then using the cost matrix, and the example-dependent cost-sensitive framework as described in Section 3.3.1, an example-dependent cost statistic is calculated as:

$$\begin{aligned} Cost_i &= y_i(c_i C_{TP_i} + (1-c_i)C_{FN_i}) \\ &\quad + (1-y_i)(c_i C_{FP_i} + (1-c_i)C_{TN_i}) \\ &= y_i(c_i(\gamma_i(C_{o_i} - CLV_i - C_a) - C_{o_i}) + CLV_i) \\ &\quad + c_i(C_{o_i} + C_a), \end{aligned} \tag{5.3}$$

leading to a total cost of:

$$Cost = \sum_{i=1}^{N} Cost_i. \tag{5.4}$$

Furthermore, using (3.7), the savings are calculated as:

$$Savings = \frac{Cost_l - Cost}{Cost_l}, \tag{5.5}$$

In almost cases the costless class ($Cost_l$) will be the negative class, as typically the distribution of a churn dataset is skewed towards the non-churners [Verbeke et al., 2012]. Given that $Cost_l$ can be expressed as $Cost(f_0)$, or simply $Cost$ with $c_i = 0$ $\forall i$:

$$Cost_l = \sum_{i=1}^{N} y_i CLV_i. \tag{5.6}$$

This is consistent with the notion that if no model is used, the total cost would be the sum of the customer lifetime values of the actual churners, which gives the insight that the $Savings$ measure is comparing the financial impact of the campaign of using a classification model against no using a model at all.

5.1.2.1 *Customer lifetime value*

Lastly, one of the key values to calculate the $Savings$, is the customer lifetime value. Within marketing there exists a common misconception between customer profitability and customer lifetime value. The two terms are usually used in an interchangeable way, creating confusion of what the actual objective of a churn modeling campaign should be. Several studies have proposed models providing a unique definition of both terms [Neslin et al., 2006; Pfeifer et al., 2004; Milne and Boza, 1999; van Raaij et al., 2003]. Customer profitability indicates the difference between the income and the cost generated by a customer i during a financial period t. It is defined as:

$$CP_{i,t} = \mu \cdot s_{i,t}, \tag{5.7}$$

where $s_{i,t}$ refers to the consumption of customer i during time period t, and μ refers to the average marginal profit by unit product usage.

Moreover, we are interested to see what is the expected income that a particular customer will generate in the future, in other words, calculating the expected sum of discount future earnings [Neslin et al., 2006]. Therefore, the CLV_i is defined as:

$$CLV_i = \sum_{t=1}^{T} \frac{\mu \cdot s_{i,t}}{(1+r)^t}, \tag{5.8}$$

where r is the discount rate, and T the number of time period. Typically T should be considered large enough since without prior knowledge a customer is expected to keep being a customer for the foreseeable future. In practice T is set up to be ∞ [Glady et al., 2009]. Also, for simplicity it can be assumed that $s_{i,t+1} = s_{i,t} \cdot (1+g)\ \forall i, t$, which means that there is a constant growth g in the customer consumption. Given that, the customer lifetime value can be re-written as

$$CLV_i = \sum_{t=1}^{\infty} \frac{(1+g)^t}{(1+r)^t} \cdot \mu \cdot s_{i,1}, \tag{5.9}$$

which in the case of $g < r$, this is a geometric series meaning that it can be expressed as

$$CLV_i = \frac{\mu \cdot s_{i,1}}{(r-g)}. \tag{5.10}$$

5.1.3 *Churn modeling database*

For the analysis we used a dataset provided by a TV cable provider. The dataset consists of active customers during the first semester of 2014. The total dataset contains 9,410 individual registries, each one with 45 attributes, including a churn label indicating whenever a customer is a churner. This label was created internally in the company, and can be regarded as highly accurate. In the dataset only 455 customers are churners, leading to a churn ratio of 4.83%.

5.1.3.1 *Offer acceptance calculation*

In practice companies have a set of offers to make to a customer as a part of the retention campaign, they vary from discounts, to upgrades among others. In the particular case of a TV cable provided, the offers include adding a new set of channels, changing the TV receiver to one with new technology (ie. high definition, video recording, 4K), or to offer a discount on the monthly bill. Unsurprisingly, not all offers apply to all clients. For instance a customer that already has all the channels can not be offered a new set of channels. Moreover, an offer usually means an additional cost to the company and not all offers have not the same cost or the same impact in reducing churn.

Taking into account the cost and the implication of the offers, the problem can be resumed in making each customer the offer that will maximize the acceptance rate and more important reducing the overall cost.

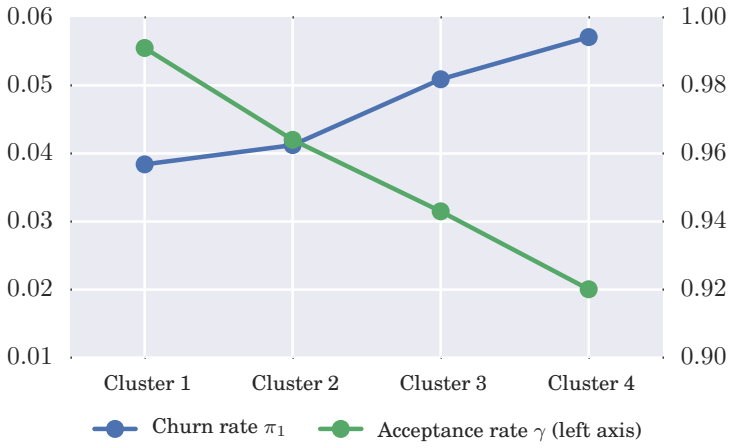


Figure 5.3: Acceptance rate (γ) of the best offer for each customer profile. As expected, the higher the churn rate the lower the acceptance rate, as it is more difficult to make a good offer to a customer which is more likely to defect.

In order to calculate the acceptance probability γ_i a champion-challenger process was made. First, the customers were grouped into clusters according to their behavioral and socio-economical characteristics. In particular the K-means algorithm was used [Marslan, 2009]. Then for a period of two months, randomly selected offers were made to the customers and their response was evaluated. Unfortunately, for confidentiality reasons we can not describe the different clusters, neither the actual offer made to each customer. Nevertheless, in Figure 5.3, the average churn rate and acceptance rate γ_i per cluster is shown. As expected, the higher the churn rate the lower the acceptance rate, as it is more difficult to make a good offer to a customer which is more likely to defect.

5.2 DIRECT MARKETING

In direct marketing the objective is to classify those customers who are more likely to have a certain response to a marketing campaign [Ngai et al., 2009]. We used a direct marketing dataset [Moro et al., 2011] available on the UCI machine learning repository [Bache and Lichman, 2013]. The dataset contains 45,000 clients of a Portuguese bank who were contacted by phone between March 2008 and October 2010 and received an offer to open a long-term deposit account with attractive interest rates. The dataset contains features such as age, job, marital

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	$C_{TP_i} = C_a$	$C_{FP_i} = C_a$
Predicted Negative $c_i = 0$	$C_{FN_i} = Int_i$	$C_{TN_i} = 0$

Table 5.2: Direct marketing example-dependent cost matrix

status, education, average yearly balance and current loan status and the label indicating whether or not the client accepted the offer.

This problem is example-dependent cost sensitive, since there are different costs of false positives and false negatives. Specifically, in direct marketing, false positives have the cost of contacting the client, and false negatives have the cost due to the loss of income by failing to contact a client that otherwise would have opened a long-term deposit.

We used the direct marketing example-dependent cost matrix proposed in [Correa Bahnsen et al., 2014b]. The cost matrix is shown in Table 5.2, where C_a is the administrative cost of contacting the client, as is credit card fraud, and Int_i is the expected income when a client opens a long-term deposit. This last term is defined as the long-term deposit amount times the interest rate spread.

In order to estimate Int_i, first the long-term deposit amount is assumed to be a 20% of the average yearly balance, and lastly, the interest rate spread is estimated to be 2.463333%, which is the average between 2008 and 2010 of the retail banking sector in Portugal as reported by the Portuguese central bank. Given that, the Int_i is equal to $(balance * 20\%) * 2.463333\%$.

Part III

PROPOSED EXAMPLE-DEPENDENT COST-SENSITIVE METHODS

6

BAYES MINIMUM RISK

Outline

In this chapter, we present our previously proposed Bayes minimum risk algorithm. The method consists in quantifying tradeoffs between various decisions using probabilities and the costs that accompany such decisions. First, in Section 6.1, we present the Bayes minimum risk algorithm. Then, in Section 6.2, we discuss the impact of calibrating the probabilities in the model. Finally, in Section 6.3, using the five real-world cost-sensitive databases, we compare the results of the proposed algorithm, against state-of-the-art methods.

6.1 BAYES MINIMUM RISK MODEL

As defined in [Jayanta K. et al., 2006], the BMR classifier is a decision model based on quantifying tradeoffs between various decisions using probabilities and the costs that accompany such decisions. This is done in a way that for each example the expected losses are minimized. In what follows, we consider the probability estimates $\hat{p}_i$ as known, regardless of the algorithm used to calculate them. The risk that accompanies each decision is calculated using the cost matrix as described in Table 3.1. In the specific framework of binary classification, the risk of predicting the example i as negative is

$$R(c_i = 0|\mathbf{x}_i) = C_{TN_i}(1 - \hat{p}_i) + C_{FN_i} \cdot \hat{p}_i, \tag{6.1}$$

and

$$R(c_i = 1|\mathbf{x}_i) = C_{TP_i} \cdot \hat{p}_i + C_{FP_i}(1 - \hat{p}_i), \tag{6.2}$$

is the risk when predicting the example as positive, where $\hat{p}_i$ is the estimated positive probability for example i. Subsequently, if

$$R(c_i = 0|\mathbf{x}_i) \leqslant R(c_i = 1|\mathbf{x}_i), \tag{6.3}$$

then the example i is classified as negative. This means that the risk associated with the decision c_i is lower than the risk associated with classifying it as positive.

CHARACTERISTICS OF PROBABILITY ESTIMATES
When using the output of a binary classifier as a basis for decision making, there is a need for a probability that not only separates well between positive and negative examples, but that also assesses the real probability of the event [Cohen and Goldszmidt, 2004].

6.2 CALIBRATION OF PROBABILITIES

In this section, two methods for calibrating probabilities are explained. First, the method proposed in [Elkan, 2001] to adjust the probabilities based on the difference in bad rates between the training and testing datasets. Then, the method proposed in Hernandez-Orallo et al. [2012], in which calibrated probabilities are extracted after modifying the ROC curve using the ROC convex hull methodology, is described.

6.2.1 *Calibration due to a change in base rates*

One of the reasons why a probability may not be calibrated is because the algorithm is trained using a dataset with a different base (or positive) rate than the one on the evaluation dataset. This is something common in machine learning since using under-sampling or over-sampling is a typical method to solve problems such as class imbalance and cost sensitivity [Hulse and Khoshgoftaar, 2007].

In order to solve this and find probabilities that are calibrated, in [Elkan, 2001] a formula that corrects the probabilities based on the difference of the base rates is proposed. The objective is using $\hat{p}$ which was estimated using a population with base rate π_1, to find $\hat{p}'$ for the real population which has a base rate π_1'. A solution for $\hat{p}'$ is given as follows:

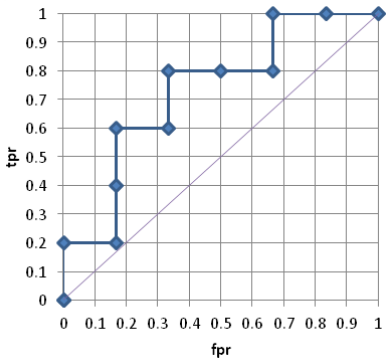
$$\hat{p}' = \pi_1' \frac{\hat{p} - \hat{p}\pi_1}{\pi_1 - \pi_1\hat{p} + \pi_1'\hat{p} - \pi_1\pi_1'}. \tag{6.4}$$

6.2.2 *Calibrated using the ROC convex hull*

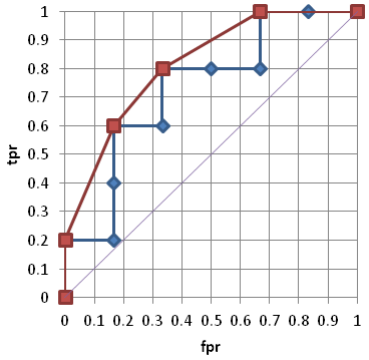
In order to illustrate the ROC convex hull approach proposed in [Hernandez-Orallo et al., 2012], let us consider the set of probabilities given in Figure 6.1a. Their corresponding ROC curve of

Probability	Label
0.0	0
0.1	1
0.2	0
0.3	0
0.4	1
0.5	0
0.6	1
0.7	1
0.8	0
0.9	1
1.0	1

(a) Set of probabilities and their respective class label



(b) ROC curve of the set of probabilities



(c) Convex hull of the ROC curve

Prob	Cal Prob
0.0	0
0.1	0.333
0.2	0.333
0.3	0.333
0.4	0.5
0.5	0.5
0.6	0.666
0.7	0.666
0.8	0.666
0.9	1
1.0	1

(d) Calibrated probabilities

Figure 6.1: Estimation of calibrated probabilities using the ROC convex hull.

that set of probabilities is shown in Figure 6.1b. It can be seen that this set of probabilities is not calibrated, since at 0.1 there is a positive example followed by 2 negative examples. That inconsistency is represented in the ROC curve as a non convex segment over the curve.

In order to obtain a set of calibrated probabilities, first the ROC curve must be modified in order to be convex. The way to do that, is to find the convex

hull [Hernandez-Orallo et al., 2012], to find the minimal convex set containing the different points of the ROC curve. In Figure 6.1c, the convex hull algorithm is applied to the previously evaluated ROC curve. It is shown that the new curve is convex, and includes all the points of the previous ROC curve.

Now that there is a new convex ROC curve or ROCCH, the calibrated probabilities can be extracted as shown in Figure 6.1d. The procedure to extract the new probabilities is to first group the probabilities according to the points in the ROCCH curve, and then make the calibrated probabilities be the slope of the ROCCH for each group.

6.3 EXPERIMENTS

In this section, first the summarized the datasets used for the experiments. Then, the partitioning of the dataset and the algorithms used are shown. Finally, we present the results.

6.3.1 *Experimental setup*

For the experiments we used five datasets from four different real world example-dependent cost-sensitive problems: Credit card fraud detection (see Section 4.1), credit scoring (see Section 4.2), churn modeling (see Section 5.1) and direct marketing (see Section 5.2).

For each dataset we used a pre-define a cost matrix that we previously proposed in different publications. Additionally, for each database, 3 different datasets are extracted: training, validation and testing. Each one containing 50%, 25% and 25% of the examples, respectively. Afterwards, because classification algorithms suffer when the label distribution is skewed towards one of the classes [Hastie et al., 2009], an under-sampling of the positive examples is made, in order to have a balanced class distribution. Moreover, we perform the cost-proportionate rejection-sampling and cost proportionate over-sampling procedures, that we previously described in Section 3.3.3. Table 6.1, summarizes the different datasets. It is important to note that the sampling procedures were only applied to the training dataset since the validation and test datasets must reflect the real distribution.

Database	**Set**	**# Obs** (N)	**%Pos** (π_1)	**Cost**
Fraud	Total	236,735	1.50	895,154
Detection	Training (t)	94,599	1.51	358,078
	Under-sampled (u)	2,828	50.42	358,078
	Rejection-sampled (r)	94,522	1.43	357,927
	Over-sampled (o)	189,115	1.46	716,006
	Validation	70,910	1.53	274,910
	Testing	71,226	1.45	262,167
Credit	Total	112,915	6.74	83,740,181
Scoring 1	Training (t)	45,264	6.75	33,360,130
	Under-sampled (u)	6,038	50.58	33,360,130
	Rejection-sampled (r)	5,271	43.81	29,009,564
	Over-sampled (o)	66,123	36.16	296,515,655
	Validation	33,919	6.68	24,786,997
	Testing	33,732	6.81	25,593,055
Credit	Total	38,969	19.88	3,117,960
Scoring 2	Training (t)	15,353	19.97	1,221,174
	Under-sampled (u)	6,188	49.56	1,221,174
	Rejection-sampled (r)	2,776	35.77	631,595
	Over-sampled (o)	33,805	33.93	6,798,282
	Validation	11,833	20.36	991,795
	Testing	11,783	19.30	904,991
Churn	Total	9,410	4.83	580,884
Modeling	Training (t)	3,758	5.05	244,542
	Under-sampled (u)	374	50.80	244,542
	Rejection-sampled (r)	428	41.35	431,428
	Over-sampled (o)	5,767	31.24	2,350,285
	Validation	2,824	4.77	174,171
	Testing	2,825	4.42	162,171
Direct	Total	37,931	12.62	59,507
Marketing	Training (t)	15,346	12.55	24,304
	Under-sampled (u)	3,806	50.60	24,304
	Rejection-sampled (r)	1,644	52.43	20,621
	Over-sampled (o)	22,625	40.69	207,978
	Validation	11,354	12.30	16,154
	Testing	11,231	13.04	19,048

Table 6.1: Summary of the datasets

6.3.2 *Results*

For the experiments we first used three classification algorithms, decision tree (DT), logistic regression (LR) and random forest (RF). Using the implementation of *Scikit-learn* [Pedregosa et al., 2011], each algorithm is trained using the different training sets: training (t), under-sampling (u), cost-proportionate rejection-sampling (r) [Zadrozny et al., 2003] and cost-proportionate over-sampling (o) [Elkan, 2001]. Afterwards, we evaluate the results of the algorithms using BMR methods. In particular, we check the impact on the results of the different calibration methods. The implementation of the cost-sensitive algorithms is done using the *CostCla* library, see Appendix A.

The results are shown in Table 6.2. First, when observing the results of the cost-insensitive methods (CI), that is, DT, LR and RF algorithms trained on the t and u sets, the RF algorithm produces the best result by savings in three out of the five sets, followed by the LR – u. It is also clear that the results on the t dataset are not as good as the ones on the u, this is highly

Family	Algorithm	Fraud	Churn	Credit 1
CI	DT-t	0.3176±0.0357	-0.0018±0.0194	0.1931±0.0087
	LR-t	0.0092±0.0002	-0.0001±0.0002	0.0177±0.0126
	RF-t	0.3342±0.0156	-0.0026±0.0079	0.1471±0.0071
	DT-u	0.5239±0.0118	-0.0389±0.0583	0.3287±0.0125
	LR-u	0.1243±0.0387	0.0039±0.0492	0.4118±0.0313
	RF-u	0.5684±0.0097	0.0433±0.0533	0.4981±0.0079
CPS	DT-r	0.3439±0.0453	0.0054±0.0568	0.3310±0.0126
	LR-r	0.3077±0.0301	0.0484±0.0375	0.3965±0.0263
	RF-r	0.3812±0.0264	0.1056±0.0412	**0.4989±0.0080**
	DT-o	0.3172±0.0274	0.0251±0.0195	0.1738±0.0092
	LR-o	0.2793±0.0185	0.0316±0.0228	0.3301±0.0109
	RF-o	0.3612±0.0295	0.0205±0.0156	0.2128±0.0081
BMR	DT-t-BMR	0.6045±0.0386	0.0226±0.0200	0.1931±0.0087
	LR-t-BMR	0.4552±0.0203	0.0872±0.0308	0.1973±0.0404
	RF-t-BMR	0.6175±0.0149	0.0435±0.0356	0.4878±0.0082
CAL	DT-t-BMR-cal	0.5936±0.0386	0.0298±0.0145	0.1054±0.0358
BMR	LR-t-BMR-cal	0.0897±0.0203	**0.1082±0.0316**	0.2189±0.0541
	RF-t-BMR-cal	**0.6414±0.0154**	0.0856±0.0354	0.4924±0.0087

(those models with the highest savings are market as bold)

Table 6.2: Results of the algorithms measured by savings

Family	Algorithm	Credit 2	Marketing
CI	DT-t	-0.0616±0.0229	-0.2342±0.0609
	LR-t	0.0039±0.0012	-0.2931±0.0602
	RF-t	0.0303±0.0040	-0.2569±0.0637
	DT-u	-0.1893±0.0314	-0.0278±0.0475
	LR-u	0.1850±0.0231	0.2200±0.0376
	RF-u	0.1237±0.0228	0.1227±0.0443
CPS	DT-r	0.0724±0.0212	0.1960±0.0527
	LR-r	0.2650±0.0115	0.4210±0.0267
	RF-r	0.3055±0.0106	0.3840±0.0360
	DT-o	0.0918±0.0225	-0.2598±0.0559
	LR-o	0.2554±0.0090	0.3129±0.0277
	RF-o	0.2242±0.0070	-0.1782±0.0618
BMR	DT-t-BMR	-0.0616±0.0230	-0.2185±0.0611
	LR-t-BMR	0.3119±0.0089	0.4915±0.0090
	RF-t-BMR	0.3027±0.0096	0.3722±0.0263
CAL	DT-t-BMR-cal	0.2740±0.0067	0.4598±0.0089
BMR	LR-t-BMR-cal	**0.3148±0.0094**	**0.4973±0.0084**
	RF-t-BMR-cal	0.3133±0.0094	0.4807±0.0093

(those models with the highest savings are market as bold)

Table 6.3: Continuation of Table 6.2.

related to the unbalanced distribution of the positives and negatives in all the databases.

In the case of cost-proportionate sampling methods (CPS), specifically the cost-proportionate rejection sampling (r) and cost-proportionate over sampling (o). It is observed than in four cases the savings increases quite significantly. It is on the fraud detection database where these methods do not outperform the algorithms trained on the under-sampled set. This may be related to the fact that in this database the initial percentage of positives is 1.5% which is similar to the percentage in the r and o sets. However it is 50.42% in the u set, which may help explain why this method performs much better as measured by savings.

Afterwards, in the case of the BMR algorithms, the results show that this method outperforms the previous ones in four cases and has almost the same result in the other set. In the fraud detection set, the results are quite better, since the savings of the three classification algorithms increase when using this methodology. Moreover, we found that by calibrating the probabilities, the results of the Bayes minimum risk increase.

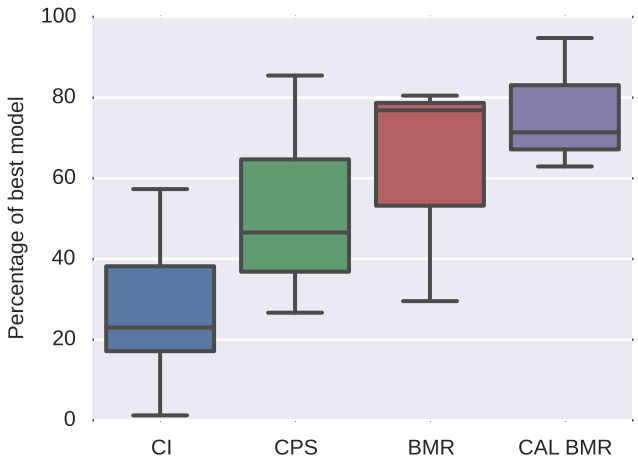


Figure 6.2: **Comparison of the average savings of the algorithms versus the highest savings by family of classifiers.** When the probabilities are calibrated there is a significant increase in savings.

In Figure 6.2, we compare the different families of algorithms, by calculating in each dataset the performance of the methods compared with the best method. First, it is shown the huge difference of using only cost-insensitive methods, compared with any of the cost-sensitive families. Furthermore, the Bayes minimum risk methods outperforms the cost-proportionate sampling methods.

Moreover, we evaluate the Brier score of the different algorithms. The objective of this, is because as mention in [Cohen and Goldszmidt, 2004], when using the output of a binary classifier as a basis for decision making, there is a need for a probability that not only separates well between positive and negative examples, but that also assesses the real probability of the event. The results are shown in Table 6.4. First of all, as expected, the logistic regression models are very well calibrated regardless of the dataset used for train them. This is because, the logistic regression algorithm is intended to estimate the most reliable probabilities as we will discuss in Chapter 7.

When comparing the results of the Bayes minimum risk with and without calibration, it is clear that the calibration of the probabilities lead to a lower Brier score as shown in Figure 6.3. Interestingly, the results of Brier score and the savings are highly correlated, confirming the intuition behind the need to calibrate the probabilities before using the Bayes minimum risk method.

Family	Algorithm	Fraud	Churn	Credit 1
CI	DT-t	0.0134±0.0005	0.0932±0.0054	0.1054±0.0016
	LR-t	0.0440±0.0008	0.0449±0.0025	0.0598±0.0016
	RF-t	0.0064±0.0002	0.0532±0.0027	0.0519±0.0007
	DT-u	0.1424±0.0071	0.4152±0.0205	0.3141±0.0052
	LR-u	0.1872±0.0195	0.2439±0.0139	0.1718±0.0088
	RF-u	0.0724±0.0007	0.2442±0.0134	0.1565±0.0019
CPS	DT-r	0.2041±0.0111	0.3595±0.0185	0.2884±0.0054
	LR-r	0.1726±0.0098	0.1906±0.0112	0.1448±0.0063
	RF-r	0.1764±0.0127	0.1964±0.0101	0.1374±0.0028
	DT-o	0.1264±0.0102	0.0963±0.0048	0.1020±0.0014
	LR-o	0.2158±0.0098	0.1136±0.0025	0.1134±0.0014
	RF-o	0.1694±0.0146	0.0701±0.0031	0.0550±0.0008
BMR	DT-t-BMR	0.0134±0.0005	0.0932±0.0054	0.1054±0.0016
	LR-t-BMR	0.0440±0.0008	0.0449±0.0025	0.0598±0.0016
	RF-t-BMR	0.0064±0.0002	0.0532±0.0027	0.0519±0.0007
CAL	DT-t-BMR-cal	0.0098±0.0004	0.0448±0.0029	0.0604±0.0009
BMR	LR-t-BMR-cal	0.0148±0.0007	**0.0444±0.0025**	0.0590±0.0018
	RF-t-BMR-cal	**0.0058±0.0003**	0.0446±0.0026	**0.0514±0.0008**

(those models with the lowest Brier score are market as bold)

Table 6.4: Results of the algorithms measured by Brier score

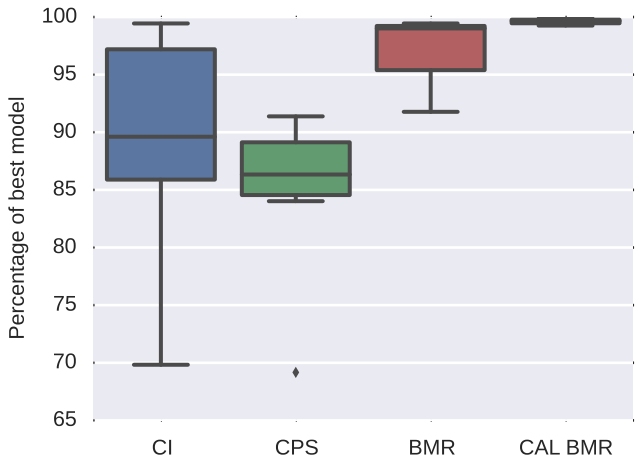


Figure 6.3: **Comparison of the average Brier score of the algorithms versus the lowest Brier score by family of classifiers.** Overall, the models that are calibrated are indeed the ones with the best Brier score.

Family	Algorithm	Credit 2	Marketing
CI	DT-t	0.3103±0.0041	0.1924±0.0035
	LR-t	0.1511±0.0016	0.0960±0.0018
	RF-t	0.1525±0.0016	0.1090±0.0018
	DT-u	0.4526±0.0057	0.4125±0.0062
	LR-u	0.2317±0.0017	0.2016±0.0027
	RF-u	0.2325±0.0024	0.2256±0.0036
CPS	DT-r	0.5516±0.0070	0.3581±0.0111
	LR-r	0.2657±0.0051	0.2029±0.0055
	RF-r	0.3613±0.0055	0.2123±0.0054
	DT-o	0.4550±0.0039	0.1904±0.0028
	LR-o	0.2258±0.0023	0.1517±0.0013
	RF-o	0.3283±0.0030	0.1190±0.0016
BMR	DT-t-BMR	0.3103±0.0041	0.1924±0.0035
	LR-t-BMR	0.1511±0.0016	0.0960±0.0018
	RF-t-BMR	0.1525±0.0016	0.1090±0.0018
CAL	DT-t-BMR-cal	0.1586±0.0019	0.1075±0.0017
BMR	LR-t-BMR-cal	**0.1505±0.0016**	**0.0952±0.0017**
	RF-t-BMR-cal	0.1510±0.0016	0.0999±0.0019

(those models with the lowest Brier score are market as bold)

Table 6.5: Continuation of Table 6.4.

7

COST-SENSITIVE LOGISTIC REGRESSION

Outline

In this chapter, we present the cost-sensitive logistic regression algorithm. The model consists in a new logistic regression cost function, one that takes into account the real costs due to misclassification and correct classification. First, in Section 7.1, we give the background behind logistic regression. Then, in Section 7.2, we described our previously proposed cost-sensitive logistic regression. For this, we do a deep analysis of the logistic regression implicit misclassification costs in Section 7.2.1. Then in Section 7.2.2, we shown a new version of the logistic regression cost function, one that takes into account the real costs due to misclassification and correct classification. Afterwards, in Section 7.2.3, we give a brief introduction to genetic algorithms, since is the model used to estimate the parameters of our proposed cost-sensitive logistic regression. Finally, in Section 7.3, we compare the results of the proposed algorithm, against state-of-the-art methods, using the five real-world cost-sensitive databases.

7.1 LOGISTIC REGRESSION

Logistic regression is a classification model that, in the specific context of binary classification, estimates the posterior probability of the positive class, as the logistic sigmoid of a linear function of the feature vector [Bishop, 2006]. The estimated probability is evaluated as

$$\hat{p}_i = P(y = 1|\mathbf{x}_i) = h_\theta(\mathbf{x}_i) = g\left(\sum_{j=1}^{k} \theta^j x_i^j\right), \tag{7.1}$$

where $h_\theta(\mathbf{x}_i)$ refers to the hypothesis of i given the parameters θ, and $g(\cdot)$ is the logistic sigmoid function, defined as

$$g(z) = \frac{1}{(1 + e^{-z})} \tag{7.2}$$

In the next figure, the logistic sigmoid function is shown.

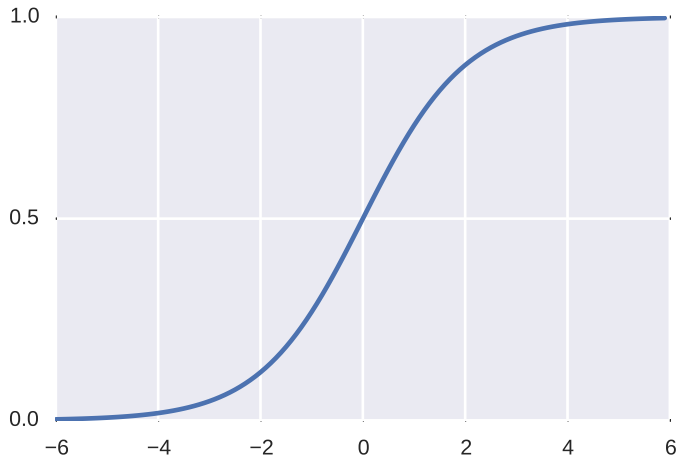


Figure 7.1: Sigmoid function

The problem then becomes on finding the right parameters that minimize a given cost function. Usually, in the case of logistic regression the cost function $J(\theta)$ refers to the negative logarithm of the likelihood, such that

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} J_i(\theta), \tag{7.3}$$

where

$$J_i(\theta) = -y_i \log(h_\theta(\mathbf{x}_i)) - (1 - y_i)\log(1 - h_\theta(\mathbf{x}_i)). \tag{7.4}$$

Therefore, the parameters are found using the following equation

$$\theta = \arg\min_{\theta} J(\theta). \tag{7.5}$$

Logistic regression estimation
There are several methods used to estimate the logistic regression, in particular, maximum likelihood [Hastie et al., 2009], Newton, coordinate descent [Murphy, 2012] and dual coordinate descent [Yu et al., 2011]. Nevertheless, all these methods rely on the assumption of convexity of the negative logarithm of the likelihood function. In the next section we analyze the convexity of the logistic regression.

7.1.1 *Convexity analysis*

A function $f(\mathbf{x})$ which is twice-differentiable is convex if and only if its hessian matrix (matrix of second-order partial deriva-

tives) is positive semi-definite [Boyd and Vandenberghe, 2010]. Therefore, with the objective of evaluating the convexity of the logistic function, first the first order partial derivatives are calculated as follows:

$$\begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \cdots \\ \frac{\partial J(\theta)}{\partial \theta_k} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N}\left[x_i^{(1)}\left(h_\theta(\mathbf{x}_i) - y_i\right)\right] \\ \frac{1}{N}\sum_{i=1}^{N}\left[x_i^{(2)}\left(h_\theta(\mathbf{x}_i) - y_i\right)\right] \\ \cdots \\ \frac{1}{N}\sum_{i=1}^{N}\left[x_i^{(k)}\left(h_\theta(\mathbf{x}_i) - y_i\right)\right] \end{bmatrix} \tag{7.6}$$

Then, using the first order partial derivatives, the Hessian (H) can be generalized as

$$\frac{\partial^2 J(\theta)}{\partial \theta_{j1}\partial \theta_{j2}} = \frac{1}{N}\sum_{i=1}^{N}\left[\left(1 - h_\theta(\mathbf{x}_i)\right)h_\theta(\mathbf{x}_i)x_i^{(j1)}x_i^{(j2)}\right], \tag{7.7}$$

where, j1 and j2 $\in \{1 \cdots k\}$. Which is the same as

$$H = \begin{bmatrix} \frac{\partial^2 J(\theta)}{\partial \theta_1 \partial \theta_1} & \frac{\partial^2 J(\theta)}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J(\theta)}{\partial \theta_1 \partial \theta_k} \\ \frac{\partial^2 J(\theta)}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J(\theta)}{\partial \theta_2 \partial \theta_2} & \cdots & \frac{\partial^2 J(\theta)}{\partial \theta_2 \partial \theta_k} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 J(\theta)}{\partial \theta_k \partial \theta_1} & \frac{\partial^2 J(\theta)}{\partial \theta_k \partial \theta_2} & \cdots & \frac{\partial^2 J(\theta)}{\partial \theta_k \partial \theta_k} \end{bmatrix} \tag{7.8}$$

For the cost function to be convex the Hessian matrix must be positive-semidefinite, and a function is positive-semidefinite if:

$$z^T\left[\nabla_x^2 f(x)\right] z \geqslant 0 \quad \forall z \tag{7.9}$$

Applied to J(θ), it is necessary to proof

$$z^T\left[H\right] z \geqslant 0 \quad \forall z \tag{7.10}$$

Moreover, H can be rewritten, using only the internal part of the sum, since it do not dependent on θ

$$H = \frac{1}{N}\sum_{i=1}^{N}\left[\left(1 - h_\theta(\mathbf{x}_i)\right)h_\theta(\mathbf{x}_i)\mathbf{x}_i^T\mathbf{x}\right]. \tag{7.11}$$

Then, (7.10) becomes

$$z^T \left[\left(1 - h_\theta(\mathbf{x}_i) \right) h_\theta(\mathbf{x}_i) \mathbf{x}_i^T \mathbf{x} \right] z, \tag{7.12}$$

which can be rewritten as

$$(1 - h_\theta(\mathbf{x}_i)) h_\theta(\mathbf{x}_i) \left(\mathbf{x}_i^T z \right)^2 \geqslant 0. \tag{7.13}$$

Therefore, proving that $J(\theta)$ is convex, as $\left(\mathbf{x}_i^T z\right)^2 \geqslant 0$ and $((1 - h_\theta(\mathbf{x}_i)) h_\theta(\mathbf{x}_i)) \geqslant 0$ because $0 \geqslant h_\theta(\mathbf{x}_i) \geqslant 1$.

7.2 COST-SENSITIVE LOGISTIC REGRESSION

In this section we introduce our proposed cost-sensitive logistic regression [Correa Bahnsen et al., 2014a]. First, we motive the need to modify the logistic regression, as we analyze the implicit costs that the logistic regression assign to each misclassification error during the estimation of the parameters θ. Then, we show our proposed algorithm.

7.2.1 *Analysis of the logistic cost function*

The logistic regression cost function, as described in (7.4), implicitly assume that a false positives and a false negatives have the same cost, i.e. $C_{FP_i} = C_{FN_i}\ \forall i \in \{1, \cdots, N\}$. This can be easily shown by analyzing the logistic cost function for both values of y_i and the algorithm prediction $h_\theta(\mathbf{x}_i)$):

- If $y_i = 0$ and $h_\theta(\mathbf{x}_i)) \approx 0$,

$$\begin{aligned} J_i(\theta) &= -y_i \log(h_\theta(\mathbf{x}_i)) - (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i)) \\ &\approx -(0) \log((0)) - (1 - (0)) \log(1 - (0)) \\ &\approx 0. \end{aligned}$$

- If $y_i = 0$ and $h_\theta(\mathbf{x}_i)) \approx 1$,

$$J_i(\theta) \approx -(0) \log((1)) - (1 - (0)) \log(1 - (1)) \approx \infty.$$

- If $y_i = 1$ and $h_\theta(\mathbf{x}_i)) \approx 0$,

$$\begin{aligned} J_i(\theta) &\approx -(1) \log((0)) - (1 - (1)) \log(1 - (0)) \\ &\approx \infty. \end{aligned}$$

- If $y_i = 1$ and $h_\theta(\mathbf{x}_i)) \approx 1$,

$$\begin{aligned} J_i(\theta) &\approx -(1)\log((1)) - (1-(1))\log(1-(1)) \\ &\approx 0. \end{aligned}$$

Then, we collect the previous results in a cost matrix:

	Actual Positive $y = 1$	Actual Negative $y = 0$
Predicted Positive $c = 1$	$C_{TP_i} \approx 0$	$C_{FP_i} \approx \infty$
Predicted Negative $c = 0$	$C_{FN_i} \approx \infty$	$C_{TN_i} \approx 0$

Table 7.1: Logistic regression cost matrix

This confirm our notion that the logistic regression cost function implicitly assume that a false positives and a false negatives have the same cost. However, as discussed in previous chapters, this is not the case in several real-world applications.

7.2.2 *Cost-sensitive cost function*

In order to incorporate the different real costs, as showed in Table 3.1, into the logistic regression. We start by analyzing the expected costs that a modified logistic regression cost function should made for each misclassification and correct classification case.

$$J_i^c(\theta) = \begin{cases} C_{TP_i} & \text{if } y_i = 1 \text{ and } h_\theta(\mathbf{x}_i) \approx 1 \\ C_{TN_i} & \text{if } y_i = 0 \text{ and } h_\theta(\mathbf{x}_i) \approx 0 \\ C_{FP_i} & \text{if } y_i = 0 \text{ and } h_\theta(\mathbf{x}_i) \approx 1 \\ C_{FN_i} & \text{if } y_i = 1 \text{ and } h_\theta(\mathbf{x}_i) \approx 0. \end{cases}$$

Then, as we already have the real costs, we create a new cost-sensitive logistic regression cost function, by including the different costs into the logistic function,

$$J^c(\theta) = \frac{1}{N}\sum_{i=1}^{N}\Big(y_i(h_\theta(\mathbf{x}_i)C_{TP_i} + (1-h_\theta(\mathbf{x}_i))C_{FN_i})$$

$$+(1-y_i)(h_\theta(\mathbf{x}_i)C_{FP_i}+(1-h_\theta(\mathbf{x}_i))C_{TN_i})\Big). \quad (7.14)$$

Nevertheless, our previous analysis have shown that this new cost function is not always convex [Correa Bahnsen et al., 2014a], therefore, we estimate the parameters θ with genetic algorithms, as this optimization heuristic does not require the underlying function to be differentiable or convex [Haupt and Haupt, 2004]. In the next section we give a brief introduction to generic algorithms.

7.2.3 *Genetic algorithms*

[TODO] Djamila: this section can: move to background, leave here, or be deleted.

Genetic algorithms, is an optimization technique that attempts to replicate natural evolution processes in which the individuals with the considered best characteristics to adapt to the environment are more likely to reproduce and survive. These advantageous individuals mate between them, producing descendants similarly characterized, so favorable characteristics are preserved and unfavorable ones destroyed, leading to the progressive evolution of the species. The model aims to improve the solution to a problem by keeping the best combination of input variables. The flow diagram presented in Figure 7.2 describes the process. It starts with the definition of the problem to optimize, generating an objective function to evaluate the possible candidate solutions (chromosomes), i.e., the objective function is the way of determining which individual produces the best outcome.

The next step is to generate an initial random population of n individuals called chromosomes that are symbolized by binary strings, where each binary position of the chromosome is called a gene and denotes a specific characteristic (input variable). Therefore the combination of all the different characteristics encoded in the string represents an individual who is a candidate for the solution. Each chromosome is evaluated in the objective function and the best individuals are selected to survive for mating (parents), while the worse ones are discarded to make room for new descendants. There are many ways of pairing the selected chromosomes [Haupt and Haupt, 2004]. In this paper, a weighted cost pairing is used, which consists of assigning a selection probability according to each chromosome

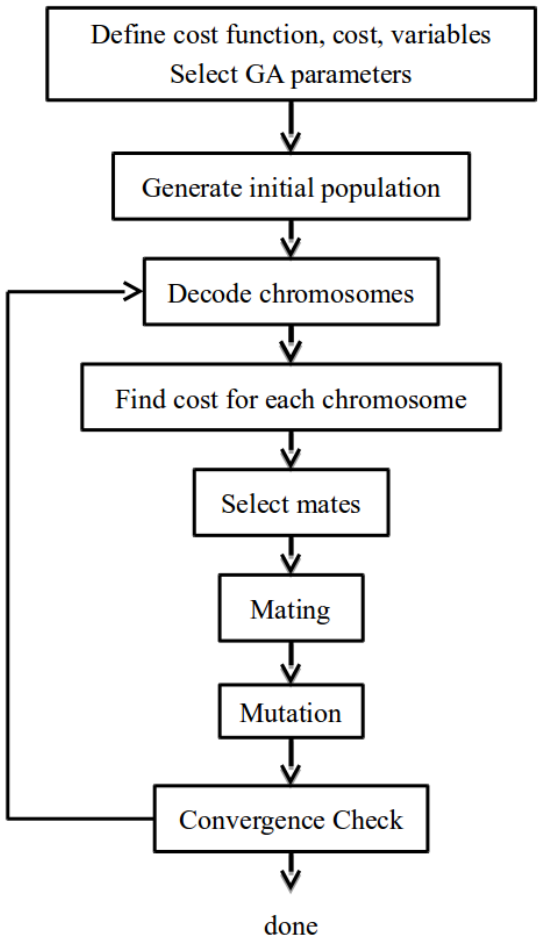


Figure 7.2: Genetic algorithms [Haupt and Haupt, 2004]

cost. That is, a chromosome with the higher cost has a greater probability of mating because cost maximization is desired.

After selecting the parent chromosomes with the chosen pairing method, the next step is to create a second generation of individuals, based on the information of the parents. There are several ways of mating. In this paper, two parents create one child. In order to transfer the parents binary information to the child, there are also different kinds of approaches such as the one-point crossover. The one-point crossover technique consists in selecting one random point on the parents string. The child is created in the following way: First, the parent1 transfers its binary code from the first gene to the crossover point. Then the parent2 transfers its binary code from the crossover point to the last gene of the chromosome. New parents are randomly selected for each new child and the process continues until the chromosome population grows back to the original size n.

Once the breeding process is completed, random mutation is used to alter a certain percentage of the genes of the chromosomes. The purpose of mutation is to introduce diversity into the population, allowing the algorithm to avoid local minima by generating new gene combinations in the chromosomes. The most common mutation procedure is the one called single point mutation. It is implemented by generating a random variable that indicates the position of the gene that will be modified, from the population of chromosomes. Generally, mutation is not allowed in the best solution chromosomes because these elite individuals are destined to propagate unchanged. In genetic algorithm this is called elitism.

Finally, after mutation is done the new generation of chromosomes is evaluated with the objective function and used in the next iteration of the described algorithm. The algorithm iterates until a maximum number of chromosome generations are created or a satisfactory solution is reached.

7.3 EXPERIMENTS

For the experiments we used five datasets from four different real world example-dependent cost-sensitive problems: Credit card fraud detection (see Section 4.1), credit scoring (see Section 4.2), churn modeling (see Section 5.1) and direct marketing (see Section 5.2). The different datasets are summarized in Table 6.1.

In particular, we are interested in compared the results of the different logistic regression models. First we trained a logisitc regression (LR) using the training (t), under-sampled (u), cost-proportionate rejection-sampling (r) [Zadrozny et al., 2003] and cost-proportionate over-sampling (o) [Elkan, 2001] datasets. Afterwards, we evaluate the results of the algorithms using BMR [Correa Bahnsen et al., 2014b]. Lastly, we calculate the cost-sensitive logistic regression (CSLR). We used the logistic regression implementation of *Scikit-learn* [Pedregosa et al., 2011], and the *CostCla* library, see Appendix A, for the cost-sensitive algorithms. Unless otherwise stated, the random selection of the training set was repeated 50 times, and in each time the models were trained and results collected, this allows us to measure the stability of the results.

In Table 7.2, we show the results of each algorithm in the different databases measured by savings. Firstly, the proposed CSLR has the highest savings in three out of the five databases,

Family	Algorithm	Fraud	Churn	Credit 1
CI	LR-t	0.0092±0.0002	-0.0001±0.0002	0.0177±0.0126
	LR-u	0.1243±0.0387	0.0039±0.0492	0.4118±0.0313
CPS	LR-r	0.3077±0.0301	0.0484±0.0375	0.3965±0.0263
	LR-o	0.2793±0.0185	0.0316±0.0228	0.3301±0.0109
BMR	LR-t-BMR	0.4552±0.0203	0.1082±0.0316	0.2189±0.0541
CST	CSLR-t	**0.6113±0.0262**	**0.1118±0.0484**	**0.4554±0.1039**

(those models with the highest savings are market as bold)

Table 7.2: Results of the algorithms measured by savings

Family	Algorithm	Credit 2	Marketing
CI	LR-t	0.0039±0.0012	-0.2931±0.0602
	LR-u	0.1850±0.0231	0.2200±0.0376
CPS	LR-r	0.2650±0.0115	0.4210±0.0267
	LR-o	0.2554±0.0090	0.3129±0.0277
BMR	LR-t-BMR	**0.3148±0.0094**	**0.4973±0.0084**
CST	CSLR-t	0.2748±0.0069	0.4484±0.0072

(those models with the highest savings are market as bold)

Table 7.3: Continuation of Table 7.2.

moreover, in the other two is the second best model. It is interesting to see how different the results from a standard logistic regression and the cost-sensitive logistic regression are. Moreover, we also calculate the results of the F_1Score for each model, as shown in Table 7.4. It is observed, that the CSLR algorithm, is not the one that gives the best results measured by F_1Score. In fact, there is not a clear relation between the results measured by savings or F_1Score.

Finally, we compute the perBest statistic for the savings and the F_1Score. This statistic is calculated as the average result

Family	Algorithm	Fraud	Churn	Credit 1
CI	LR-t	0.1531±0.0045	0.0000±0.0000	0.0494±0.0277
	LR-u	0.0241±0.0163	0.1222±0.0098	0.3160±0.0314
CPS	LR-r	0.1846±0.0123	0.1258±0.0111	0.3597±0.0156
	LR-o	0.1776±0.0117	0.1085±0.0203	**0.3769±0.0067**
BMR	LR-t-BMR	0.1384±0.0044	**0.1370±0.0150**	0.1915±0.0340
CST	CSLR-t	**0.2031±0.0065**	0.1134±0.0151	0.1454±0.0517

(those models with the highest F_1Score are market as bold)

Table 7.4: Results of the algorithms measured by F_1Score

Family	Algorithm	Credit 2	Marketing
CI	LR-t	0.0155±0.0037	0.2702±0.0125
	LR-u	**0.3890±0.0053**	0.3440±0.0083
CPS	LR-r	0.3793±0.0049	0.3374±0.0101
	LR-o	0.3804±0.0044	**0.3568±0.0102**
BMR	LR-t-BMR	0.3572±0.0045	0.2954±0.0079
CST	CSLR-t	0.3363±0.0045	0.2339±0.0051

(those models with the highest F_1 Score are market as bold)

Table 7.5: Continuation of Table 7.4.

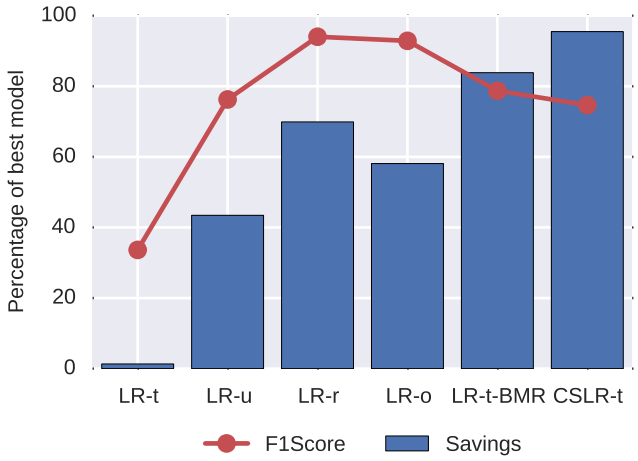


Figure 7.3: **Comparison of the average savings and F_1 Score of the algorithms versus the the best model.** The models that perform the best measured by F_1 Score are not the best in terms of savings.

of each algorithm compared with the best in each set. The results are shown in Figure 7.3. In average, the CSLR is the best model, as it arises to 95.5% of the best model in the different databases. Nevertheless, when measured by F_1 Score, it only arises to 74.7% of the best model. This, leads to the conclusion of the importance of using a cost-sensitive measure such as the savings, when evaluating real-world example-dependent cost-sensitive problems.

Moreover, it is observed that the standard logistic regression trained using the training set, is the model that performs the worst measured by both savings and F_1 Score. This is related not only to the cost-sensitivity of the problems, but also to the highly unbalanced distribution of positive and negatives presented in all the databases. This is the reason why by using an

under-sampling procedure, the results are improved by both measures.

8

COST-SENSITIVE DECISION TREES

Outline

In this chapter, we introduce our proposed cost-sensitive decision tree algorithm. The algorithm is based in incorporating the different example-dependent costs into a new cost-based impurity measure and a new cost-based pruning criteria. First, in Section 8.1, we give the background behind the decision tree algorithm. Then, in Section 8.2, we introduce the cost-sensitive decision tree method. For this, we first shown the new cost-sensitive impurity measure, that introduces the diferent costs when evaluating the performance of a split. Then, we introduce a new cost-sensitive pruning criteria. Finally, in Section 8.3, we compare the results of the proposed algorithm, against state-of-the-art methods, using the five real-world cost-sensitive databases.

8.1 DECISION TREES

Decision trees are one of the most widely used machine learning algorithms [Maimon, 2008]. The technique is considered to be white box, in the sense that is easy to interpret, and has a very low computational cost, while maintaining a good performance as compared with more complex techniques [Hastie et al., 2009]. There are two types of decision tree depending on the objective of the model. They work either for classification or regression. In this section we focus on binary classification decision tree.

8.1.1 *Construction of classification trees*

Classification trees is one of the most common types of decision tree, in which the objective is to find the $Tree$ that best discriminates between classes. In general the decision tree represents a set of splitting rules organized in levels in a flowchart structure.

8.1.1.1 *Splitting criteria*

In the Tree, each rule is shown as a node, and it is represented as $(\mathbf{x}^j, l^j)$, meaning that the set $\mathcal{S}$ is split in two sets $\mathcal{S}^l$ and $\mathcal{S}^r$ according to $\mathbf{x}^j$ and l^j:

$$\mathcal{S}^l = \{\mathbf{x}_i^* | \mathbf{x}_i^* \in \mathcal{S} \wedge x_i^j \leqslant l^j\}, \tag{8.1}$$

and

$$\mathcal{S}^r = \{\mathbf{x}_i^* | \mathbf{x}_i^* \in \mathcal{S} \wedge x_i^j > l^j\}, \tag{8.2}$$

where $\mathbf{x}^j$ is the j^{th} feature represented in the vector $\mathbf{x}^j = [x_1^j, x_2^j, ..., x_N^j]$, and l^j is a value such that $min(\mathbf{x}^j) \leqslant l^j < max(\mathbf{x}^j)$. Moreover, $\mathcal{S} = \mathcal{S}^l \cup \mathcal{S}^r$.

After the training set have been split, the percentage of positives in the different sets is calculated. First, the number of positives in each set is estimated by

$$\mathcal{S}_1 = \{\mathbf{x}_i^* | \mathbf{x}_i^* \in \mathcal{S} \wedge y_i = 1\}, \tag{8.3}$$

then the percentage of positives is calculates as

$$\pi_1 = |\mathcal{S}_1|/|\mathcal{S}|. \tag{8.4}$$

Then, the impurity of each leaf is calculated using either a misclassification error, entropy or Gini measures:

- Misclassification

$$I_m(\pi_1) = 1 - max(\pi_1, 1 - \pi_1) \tag{8.5}$$

- Entropy

$$I_e(\pi_1) = -\pi_1 \log \pi_1 - (1 - \pi_1) \log(1 - \pi_1) \tag{8.6}$$

- Gini

$$I_g(\pi_1) = 2\pi_1(1 - \pi_1) \tag{8.7}$$

In Figure 8.1, the three measures are shown. All measures are similar, with the exception that the gini index and the cross-entropy are differentiable, which makes them more suitable for numerical optimization.

Finally the gain of the splitting criteria using the rule $(\mathbf{x}^j, l^j)$ is calculated as the impurity of $\mathcal{S}$ minus the weighted impurity of each leaf:

$$Gain(\mathbf{x}^j, l^j) = I(\pi_1) - \frac{|\mathcal{S}^l|}{|\mathcal{S}|} I(\pi_1^l) - \frac{|\mathcal{S}^r|}{|\mathcal{S}|} I(\pi_1^r), \tag{8.8}$$

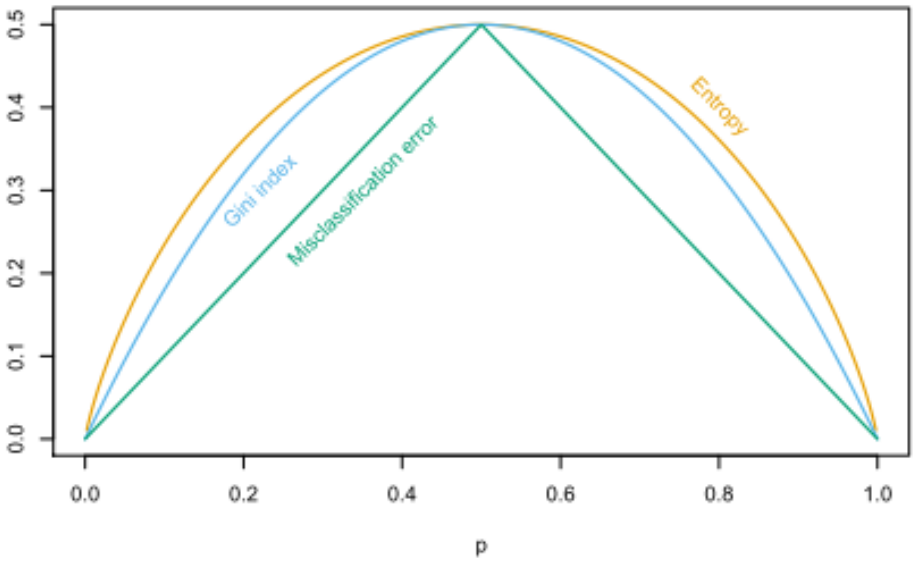


Figure 8.1: Impurity measures for a binary classification, as a function of the proportion of positive examples in the set (Cross-entropy is scaled) [Hastie et al., 2009].

where $I(\pi_1)$ can be either of the impurity measures $I_e(\pi_1)$ or $I_g(\pi_1)$.

Subsequently, the gain of all possible splitting rules is calculated. The rule with maximal gain is selected

$$(best_x, best_l) = \arg\max_{(x^j, l^j)} Gain(x^j, l^j), \tag{8.9}$$

and the set S is split into S^l and S^r according to that rule.

8.1.1.2 *Tree growing*

In order to grow a tree typical algorithms use a top-down induction using a greedy search in each iteration [Rokach and Maimon, 2010]. In each iteration, the algorithms evaluates all possible splitting rules and pick the one that maximizes the splitting criteria. After the selection of a splitting rule, each leaf is further selected and it is subdivides into smaller leafs, until one of the stopping criteria is meet. In Algorithm 8.1, the pseudocode of the tree growing procedure is presented.

Algorithm 8.1. *Pseudocode of the tree growing procedure.*

```
function TreeGrow(S)
    Tree = Empty
    if Stopping(Tree, S) then
        return Tree
    end if
    # For all features
    for j = 1 to k do
        # For all possible thresholds
        for m = 1 to N_k do
            Gains(j, m) = Gain(x^j, l^j_m, S)
        end for
    end for
    # Select the threshold with the highest gain
    (j*, l*) = argmax_(j,m) Gains
    # Split the set using the best rule
    S^l = {x*_i | x*_i ∈ S ∧ x^j*_i ⩽ l*}
    S^r = {x*_i | x*_i ∈ S ∧ x^j*_i > l*}
    # Recursively continue growing tree
    TreeGrow(S^l)
    TreeGrow(S^r)
    # Return the tree
    return Tree
end function
```

8.1.1.3 *Stopping criteria*

As the growing phase of the algorithm continue, at each iteration the stopping criteria are evaluated [Rokach and Maimon, 2010]. The most common stopping criteria are:

- All examples in the set belong to the same class. Meaning that either $\pi_1 = 1$ or $\pi_1 = 0$.
- The maximum number of iterations have been reached.
- The number of examples in S are less than the minimum number of examples defined for a split.
- The number of examples in S^l or S^r are less than the minimum number of examples defined for a leaf.
- The best splitting rule has a Gain lower than a defined threshold.

8.1.1.4 *Pruning techniques*

After a decision tree has been fully grown, there is the big chance that the algorithm generates a large tree that is probably over fitting the training data. In order to solve this in [Breiman et al., 1984] originally suggest the use of pruning techniques after the tree growing phase. The overall objective of pruning is to eliminate branches that are not contributing to the generalization accuracy of the tree [Rokach and Maimon, 2010].

In general pruning techniques start from a fully grown tree, and recursively check if by eliminating a Branch there is an Improvement in the error rate Err of the Tree. There are two main methods to calculate the Improvement of the error rate Err of the Tree, cost-complexity and error based pruning:

- Cost-complexity pruning:

  Initially proposed by Breiman [Breiman et al., 1984], this method evaluate iterative if the removal of a Branch improve the error rate Err of a Tree, weighted by the difference of the number of leafs trees.

$$PC_{cc} = \frac{Err(EB(Tree, branch), S) - Err(Tree, S)}{|Tree| - |EB(Tree, branch)|} \quad (8.10)$$

  where $Err(Tree, S)$ calculate the error rate of the Tree in set S and $EB(Tree, branch)$ is an auxiliary function that removes branch from Tree and return a new Tree. At each iteration, the current Tree is compared against all possible Branches.

- Error based pruning:

  This method proposed by Quinlan [Quinlan, 1992] is more pessimistic and instead of using the error rate Err calculate the upper bound of the confidence interval of the error Err_{UB} calculated assuming a normal distribution with a defined significance level Z_α.

$$PC_{eb} = \frac{Err_{UB}(EB(Tree, branch), S) - Err_{UB}(Tree, S)}{|Tree| - |EB(Tree, branch)|} \quad (8.11)$$

where

$$\mathrm{Err}_{\mathrm{UB}}(\mathrm{Tree},\mathcal{S}) = \mathrm{Err}(\mathrm{Tree},\mathcal{S}) + Z_\alpha \sqrt{\frac{\mathrm{Err}(\mathrm{Tree},\mathcal{S})(1-\mathrm{Err}(\mathrm{Tree},\mathcal{S}))}{|\mathcal{S}|}} \tag{8.12}$$

Moreover, as a difference from cost-complexity pruning, in the error based method, at each iteration not all the branches are compared, but only the ones on the same level. Meaning leafs are only compared against other leafs that are in the same level of iterations when the Tree was constructed. This allows for much faster pruning procedure.

In Algorithm 8.2, the general method of pruning is shown. The pruning procedure consists in evaluating the Improvement of eliminating all possible branches in a Tree, and then eliminate the one with the higher Improvement and repeat the process until a stopping criteria is meet.

Algorithm 8.2. *Pseudocode of the tree pruning procedure.*

```
function TreePruning(S, Tree)
    # For all branches
    for branch in Tree.branches do
        # Evaluate the pruning criteria
        Improvements(branch) = PC(S, Tree, branch)
    end for
    # Select the branch to prune
    branch* = argmax_branch Improvements
    # Prune selected branch
    Tree = EB(TreeTree, branch*)
    # Recursively continue pruning the tree
    TreePruning(S, Tree)
    return Tree
end function
```

8.1.1.5 *Categorical features*

When using continuous features the splitting is done by testing all possible thresholds on the database and selecting the one that maximizes the desired measure. Nevertheless, this is not straight forward when using a categorical feature, since testing all possible ways of binning the feature becomes prohibitively

time consuming the feature has many categories. Instead, a common method for doing this is to calculate π_1 of each category, then sort it, and applied the method as a continuous feature [Marslan, 2009].

DECISION TREE ALGORITHMS

Two main branches of decision trees has being studied in the last years. The main difference is the impurity measure used when splitting. First the CART algorithm that is based in the Gini index, and later the ID3 and C4.5 which uses the entropy measure.

- CART or classification and regression trees was introduced by Brieman et al. in 1984 [Breiman et al., 1984]. It is based on using the Gini index as the impurity measure and the tree is grow until all examples in each leaf belong to the same class. Afterwards, the tree is pruned using the cost-complexity method [Rokach and Maimon, 2010; Marslan, 2009].
- ID3 algorithm uses entropy as the impurity measure. The growing of the tree stop when all examples belong of each leaf belongs to the same class. In ID3 no pruning is applied [Quinlan, 1992].
- C4.5 the extension of ID3 both proposed by Quinlan [Quinlan, 1992]. Both are similar regarding the measure used, but C4.5 define the stopping criteria during the growth process to be when the number of examples in a set is less than a threshold. Moreover, after the tree is created a error based pruning is applied [Rokach and Maimon, 2010].

8.2 EXAMPLE-DEPENDENT COST-SENSITIVE DECISION TREES

Standard decision tree algorithms focus on inducing trees that maximize accuracy. However this is not optimal when the misclassification costs are unequal [Elkan, 2001]. This has led to many studies that develop algorithms that aim to introduce the cost-sensitivity into the algorithms [Lomax and Vadera, 2013]. These studies have focused on introducing the class-dependent

costs [Draper et al., 1994; Ting, 2002; Ling et al., 2004; Li et al., 2005; Kretowski and Grześ, 2006; Vadera, 2010], which is not optimal for some applications. For example in credit card fraud detection, it is true that false positives have a different cost than false negatives, nevertheless, false negatives may vary significantly, which makes class-dependent cost-sensitive methods not suitable for this problem.

In this section, we first propose a new method to introduce the costs into the decision tree induction stage, by creating new-cost based impurity measures. Afterwards, we propose a new pruning method based on minimizing the cost as pruning criteria.

8.2.1 *Cost-sensitive impurity measures*

Standard impurity measures such as misclassification, entropy or Gini, take into account the distribution of classes of each leaf to evaluate the predictive power of a splitting rule, leading to an impurity measure that is based on minimizing the misclassification rate. However, as has been previously shown [Correa Bahnsen et al., 2013], minimizing misclassification does not lead to the same results than minimizing cost. Instead, we are interested in measuring how good is a splitting rule in terms of cost not only accuracy. For doing that, we propose a new example-dependent cost based impurity measure that takes into account the cost matrix of each example.

We define a new cost-based impurity measure taking into account the costs when all the examples in a leaf are classified both as negative using f_0 and positive using f_1

$$I_c(\mathcal{S}) = \min\left\{Cost(f_0(\mathcal{S})), Cost(f_1(\mathcal{S}))\right\}. \tag{8.13}$$

The objective of this measure is to evaluate the lowest expected cost of a splitting rule. Following the same logic, the classification of each set is calculated as the prediction that leads to the lowest cost

$$f(\mathcal{S}) = \begin{cases} 0 & \text{if} \quad Cost(f_0(\mathcal{S})) \leqslant Cost(f_1(\mathcal{S})) \\ 1 & \text{otherwise} \end{cases} \tag{8.14}$$

Finally, using the cost-based impurity, the splitting criteria cost based gain of using the splitting rule (x^j, v^j) is calculated with (8.8).

8.2.2 *Cost-sensitive pruning*

Most of the literature in class-dependent cost-sensitive decision tree focuses on using the misclassification costs during the construction of the algorithms [Lomax and Vadera, 2013]. Only few algorithms such as AUCSplit [Ferri et al., 2002] have included the costs both during and after the construction of the tree. However, this approach only used the class-dependent costs, and not the example-dependent costs.

We propose a new example-dependent cost-based impurity measure, by replacing the error rate Err in (8.10) with the cost of using the Tree on $\mathcal{S}$ i.e. by replacing with $\mathrm{Cost}(f(\mathcal{S}))$.

$$\mathrm{PC}_c = \frac{\mathrm{Cost}(f(\mathcal{S})) - \mathrm{Cost}(f^*(\mathcal{S}))}{|\mathrm{Tree}| - |\mathrm{EB}(\mathrm{Tree}, node)|}, \tag{8.15}$$

where f^* is the classifier of the tree without the selected node $\mathrm{EB}(\mathrm{Tree}, node)$.

Using the new pruning criteria, nodes of the tree that do not contribute to the minimization of the cost will be pruned, regardless of the impact of those nodes on the accuracy of the algorithm. This follows the same logic as in the proposed cost-based impurity measure, since minimizing the misclassification is different than minimizing the cost, and in several real-world applications the objectives align with the cost not with the misclassification error.

8.3 EXPERIMENTS

In this section we present the experimental results. First, we evaluate the performance of the proposed CSDT algorithm and compare it against a classical decision tree (DT). We evaluate the different trees using them without pruning ($notp$), with error based pruning ($errp$), and also with the proposed cost-sensitive pruning technique ($costp$). The different algorithms are trained using the training (t), under-sampling (u), cost-proportionate rejection-sampling (r), and cost-proportionate over-sampling (o) datasets. We used the decision trees implementation of *Scikit-learn* [Pedregosa et al., 2011], and the *CostCla* library, see Appendix A, for the cost-sensitive algorithms.

For the experiments we used three datasets from three different real world example-dependent cost-sensitive problems: Credit card fraud detection (see Section 4.1), credit scoring (see

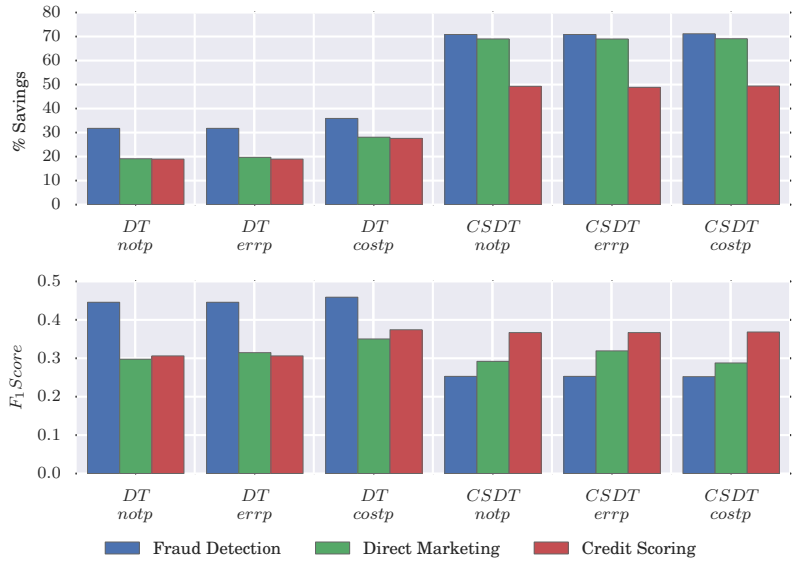


Figure 8.2: Results of the DT and the CSDT. For both algorithms, the results are calculated with and without both types of pruning criteria.

Section 4.2) and direct marketing (see Section 5.2). The different datasets are summarized in Table 6.1.

We evaluate a decision tree constructed using the Gini impurity measure, with and without the pruning defined in (8.10). We also apply the cost-based pruning procedure given in (8.15). Lastly, we compared against the proposed CSDT constructed using the cost-based impurity measure defined in (8.13), using the two pruning procedures.

In Figure 8.2, the results using the three databases are shown. In particular we first evaluate the impact of the algorithms when trained using the training set. There is a clear difference between the savings of the DT and the CSDT algorithms. However, that difference is not observable on the F_1Score results. Since the CSDT is focused on maximizing the savings not the accuracy or F_1Score. There is a small increase in savings when using the DT with cost-sensitive pruning. Nevertheless, in the case of the CSDT algorithm, there is no change when using any pruning procedure, neither in savings or F_1Score.

In addition, we also evaluate the algorithms on the different sets, under-sampling, rejection-sampling and over-sampling. The results are shown in Table 8.1. Moreover, in Figure 8.3, the average results of the different algorithms measured by savings is shown. The best results are found when using the training set. When using the under-sampling set there is a decrease in

		Fraud Detection		Direct Marketing		Credit Scoring	
set	Algorithm	%Sav	F_1 Score	%Sav	F_1 Score	%Sav	F_1 Score
t	DT_{notp}	31.76	0.4458	19.11	0.2976	18.95	0.3062
	DT_{errp}	31.76	0.4458	19.70	0.3147	18.95	0.3062
	DT_{costp}	35.89	0.4590	28.08	0.3503	27.59	0.3743
	$CSDT_{notp}$	70.85	0.2529	69.00	0.2920	49.28	0.3669
	$CSDT_{errp}$	70.85	0.2529	68.97	0.3193	48.85	0.3669
	$CSDT_{costp}$	71.16	0.2522	69.105	0.2878	49.39	0.3684
u	DT_{notp}	52.39	0.1502	49.80	0.3374	48.91	0.2983
	DT_{errp}	52.39	0.1502	49.80	0.3374	48.91	0.2983
	DT_{costp}	70.26	0.2333	53.20	0.3565	49.77	0.3286
	$CSDT_{notp}$	12.46	0.0761	64.21	0.2830	30.68	0.2061
	$CSDT_{errp}$	14.98	0.0741	66.21	0.2822	41.49	0.2564
	$CSDT_{costp}$	15.01	0.0743	68.07	0.2649	44.89	0.2881
r	DT_{notp}	34.39	0.4321	68.59	0.3135	48.97	0.3931
	DT_{errp}	34.39	0.4321	68.79	0.3196	48.97	0.3931
	DT_{costp}	38.99	0.4478	69.27	0.3274	50.48	0.3501
	$CSDT_{notp}$	70.85	0.2529	66.87	0.2761	31.25	0.1940
	$CSDT_{errp}$	70.85	0.2529	67.47	0.2581	40.69	0.2529
	$CSDT_{costp}$	71.09	0.2515	68.08	0.2642	44.51	0.2869
o	DT_{notp}	31.72	0.4495	60.30	0.3674	47.39	0.3994
	DT_{errp}	31.72	0.4495	60.30	0.3674	47.39	0.3994
	DT_{costp}	37.35	0.4575	69.44	0.3108	50.92	0.3977
	$CSDT_{notp}$	70.84	0.2529	68.75	0.2935	41.37	0.2205
	$CSDT_{errp}$	70.84	0.2529	68.75	0.2935	41.38	0.3457
	$CSDT_{costp}$	71.09	0.2515	68.75	0.2935	41.65	0.2896

Table 8.1: Results on the three datasets of the cost-sensitive and standard decision tree, without pruning (notp), with error based pruning (errp), and with cost-sensitive pruning technique (costp). Estimated using the different training sets: training, under-sampling, cost-proportionate rejection-sampling and cost-proportionate over-sampling

Figure 8.3: Average savings on the three datasets of the different cost-sensitive and standard decision tree, estimated using the different training sets: training, under-sampling, cost-proportionate rejection-sampling and cost-proportionate over-sampling.

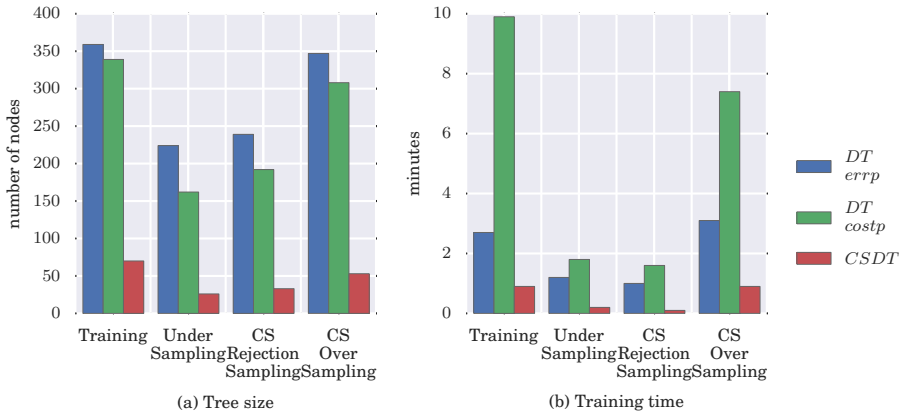


Figure 8.4: Average tree size (a) and training time (b), of the different cost-sensitive and standard decision tree, estimated using the different training sets.

savings of the CSDT algorithm. Lastly, in the case of the cost-proportionate sampling sets, there is a small increase in savings when using the CSDT algorithm.

Finally, we also analyze the different models taking into account the complexity and the training time. In particular we evaluate the size of each Tree. In Table 8.2, and Figure 8.4, the results are shown. The CSDT algorithm creates significantly smaller trees, which leads to a lower training time. In particular this is a result of using the non weighted gain, the CSDT only accepts splitting rules that contribute to the overall reduction of the cost, which is not the case if instead the weighted gain was used. Even that the DT with cost pruning, produce a good result measured by savings, it is the one that takes the longer to estimate. Since the algorithm first creates a big deci-

sion tree using the Gini impurity, and then attempt to find a smaller tree taking into account the cost. Measured by training time, the CSDT is by all means faster to train than the DT algorithm, leading to an algorithm that not only gives better results measured by savings but also one that can be trained much quicker than the standard DT.

		Fraud Detection		Direct Marketing		Credit Scoring	
set	Algorithm	|Tree|	Time	|Tree|	Time	|Tree|	Time
t	DT_{notp}	488	2.45	298	1.58	292	1.58
	DT_{errp}	488	3.90	298	2.13	292	2.13
	DT_{costp}	446	19.19	291	5.23	280	5.23
	$CSDT_{notp}$	89	1.47	51	0.40	69	0.40
	$CSDT_{errp}$	88	1.87	51	0.64	69	0.64
	$CSDT_{costp}$	89	1.74	51	0.45	69	0.45
u	DT_{notp}	308	1.10	198	1.00	167	1.00
	DT_{errp}	308	1.43	198	1.14	167	1.14
	DT_{costp}	153	2.59	190	1.34	142	1.34
	$CSDT_{notp}$	14	0.20	23	0.17	42	0.17
	$CSDT_{errp}$	14	0.23	23	0.19	42	0.19
	$CSDT_{costp}$	14	0.24	23	0.18	42	0.18
r	DT_{notp}	268	0.98	181	0.90	267	0.90
	DT_{errp}	268	1.24	181	0.95	267	0.95
	DT_{costp}	153	2.48	162	1.20	261	1.20
	$CSDT_{notp}$	18	0.22	10	0.07	70	0.07
	$CSDT_{errp}$	18	0.23	10	0.07	70	0.07
	$CSDT_{costp}$	18	0.23	10	0.07	70	0.07
o	DT_{notp}	425	2.30	340	1.80	277	1.80
	DT_{errp}	425	3.98	340	2.65	277	2.65
	DT_{costp}	364	10.15	288	5.99	273	5.99
	$CSDT_{notp}$	37	1.58	51	0.38	70	0.38
	$CSDT_{errp}$	37	1.90	51	0.45	70	0.45
	$CSDT_{costp}$	37	1.98	51	0.42	70	0.42

Table 8.2: Training time and tree size of the different cost-sensitive and standard decision tree, estimated using the different training sets: training, under-sampling, cost-proportionate rejection-sampling and cost-proportionate over-sampling, for the three databases.

9 ENSEMBLES OF COST-SENSITIVE DECISION TREES

Outline

In this chapter, we introduce the framework of ensembles of example-dependent cost-sensitive decision-trees, by training example-dependent cost-sensitive decision trees using four different random inducer methods and then blending them using three different combination approaches. First, in Section 9.1, we give the background behind ensemble learning. Then, in Section 9.2, we present our previously proposed ensembles of cost-sensitive decision-trees framework. Moreover, in Section 9.3, we prove theoretically that combining individual cost-sensitive classifiers achieves better results in the sense of higher financial savings. Finally, in Section 9.4, we compare the results of the proposed algorithm, against state-of-the-art methods, using the five real-world cost-sensitive databases.

9.1 ENSEMBLE METHODS

Ensemble learning is a widely studied topic in the machine learning community. The main idea behind the ensemble methodology is to combine several individual base classifiers in order to have a classifier that outperforms each of them [Rokach, 2009]. Nowadays, ensemble methods are one of the most popular and well studied machine learning techniques [Zhou, 2012], and it can be noted that since 2009 all the first-place and second-place winners of the KDD-Cup competition[1] used ensemble methods. The core principle in ensemble learning, is to induce random perturbations into the learning procedure in order to produce several different base classifiers from a single training set, then combining the base classifiers in order to make the final prediction. In order to induce the random permutations and therefore create the different base classifiers, several methods have been proposed, in particular: bagging [Breiman, 1996], pasting [Breiman, 1999], random forests [Breiman, 2001] and random patches [Louppe and

1 https://www.sigkdd.org/kddcup/

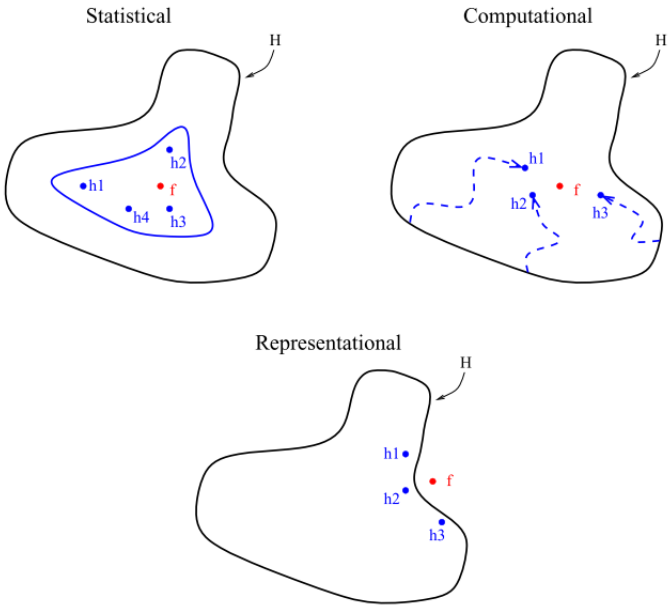


Figure 9.1: Main reasons regarding why ensemble methods perform better than single models: statistical, computational and representational [Dietterich, 2000].

Geurts, 2012]. Finally, after the base classifiers are trained, they are typically combined using either majority voting, weighted voting or stacking [Zhou, 2012].

As shown in Figure 9.1, there are three main reasons regarding why ensemble methods perform better than single models: statistical, computational and representational [Dietterich, 2000]. First, from a statistical point of view, when the learning set is too small, an algorithm can find several good models within the search space, that arise to the same performance on the training set $\mathcal{S}$. Nevertheless, without a validation set, there is a risk of choosing the wrong model. The second reason is computational; in general, algorithms rely on some local search optimization and may get stuck in a local optima. Then, an ensemble may solve this by focusing different algorithms to different spaces across the training set. The last reason is representational. In most cases, for a learning set of finite size, the true function f cannot be represented by any of the candidate models. By combining several models in an ensemble, it may be possible to obtain a model with a larger coverage across the space of representable functions.

The most typical form of an ensemble is made by combining T different base classifiers. Each base classifier $M(\mathcal{S}_j)$ is trained by applying algorithm M to a random subset $\mathcal{S}_j$ of the training

set $\mathcal{S}$. For simplicity we define $M_j \equiv M(\mathcal{S}_j)$ for $j = 1, \ldots, T$, and $\mathcal{M} = \{M_j\}_{j=1}^{T}$ a set of base classifiers. Then, these models are combined using majority voting to create the ensemble H as follows

$$f_{mv}(\mathcal{S}, \mathcal{M}) = \arg\max_{c \in \{0,1\}} \sum_{j=1}^{T} \mathbf{1}_c(M_j(\mathcal{S})). \tag{9.1}$$

THEORETICAL PERFORMANCE OF AN ENSEMBLE

If we assume that each one of the T base classifiers has a probability ρ of being correct, the probability of an ensemble making the correct decision, denoted by P_c, can be calculated using the binomial distribution [Hansen and Salamon, 1990]

$$P_c = \sum_{j>T/2}^{T} \binom{T}{j} \rho^j (1-\rho)^{T-j}. \tag{9.2}$$

Furthermore, as shown in Lam and Suen [1997], if $T \geqslant 3$ then:

$$\lim_{T\to\infty} P_c = \begin{cases} 1 & \text{if } \rho > 0.5 \\ 0 & \text{if } \rho < 0.5 \\ 0.5 & \text{if } \rho = 0.5, \end{cases} \tag{9.3}$$

leading to the conclusion that

$$\rho \geqslant 0.5 \quad \text{and} \quad T \geqslant 3 \quad \Rightarrow \quad P_c \geqslant \rho. \tag{9.4}$$

9.1.1 *Cost-sensitive ensembles*

In the context of cost-sensitive classification, some authors have proposed methods for using ensemble techniques. In [Masnadi-shirazi and Vasconcelos, 2011], the authors proposed a framework for cost-sensitive boosting that is expected to minimized the losses by using optimal cost-sensitive decision rules. In [Street, 2008], a bagging algorithm with adaptive costs was proposed. In his doctoral thesis, Nesbitt [Nesbitt, 2010], proposed a method for cost-sensitive tree-stacking. In this method different decision trees are learned, and then combined in a way that a cost function is minimize. Lastly in [Lomax and Vadera, 2013], a survey of application of cost-sensitive learning with decision trees is shown, in particular including other methods that cre-

ate cost-sensitive ensembles. However, in all these methods, the misclassification costs only dependent on the class, therefore, assuming a constant cost across examples. As a consequence, these methods are not well suited for example-dependent cost-sensitive problems.

9.2 ENSEMBLES OF COST-SENSITIVE DECISION TREES

In this section we shown our previously proposed framework of ensembles of example-dependent cost-sensitive decision-trees [Correa Bahnsen et al., 2015c], by training example-dependent cost-sensitive decision trees using four different random inducer methods and then blending them using three different combination approaches. Moreover, we propose two new cost-sensitive combination approaches, cost-sensitive weighted voting and cost-sensitive stacking. The latter being an extension of our previously proposed cost-sensitive logistic regression [Correa Bahnsen et al., 2014a].

The remainder of the section is organized as follows: First, we introduce the different random inducers used to create the base classifiers. Then, we present the combination methods. Finally, we define our proposed algorithms.

9.2.1 *Random inducers*

With the objective of creating an ensemble of example-dependent cost-sensitive decision trees, we first create T different random subsamples $\mathcal{S}_j$ for $j = 1, \ldots, T$, of the training set $\mathcal{S}$, and train a CSDT algorithm on each one. In particular we create the different subsets using four different methods: bagging [Breiman, 1996], pasting [Breiman, 1999], random forests [Breiman, 2001] and random patches [Louppe and Geurts, 2012].

In bagging [Breiman, 1996], base classifiers are built on randomly drawn bootstrap subsets of the original data, hence producing different base classifiers. Similarly, in pasting [Breiman, 1999], the base classifiers are built on random samples without replacement from the training set. In random forests [Breiman, 2001], using decision trees as the base learner, bagging is extended and combined with a randomization of the input features that are used when considering candidates to split internal nodes. In particular, instead of looking for the best split among all features, the algorithm selects, at each node, a ran-

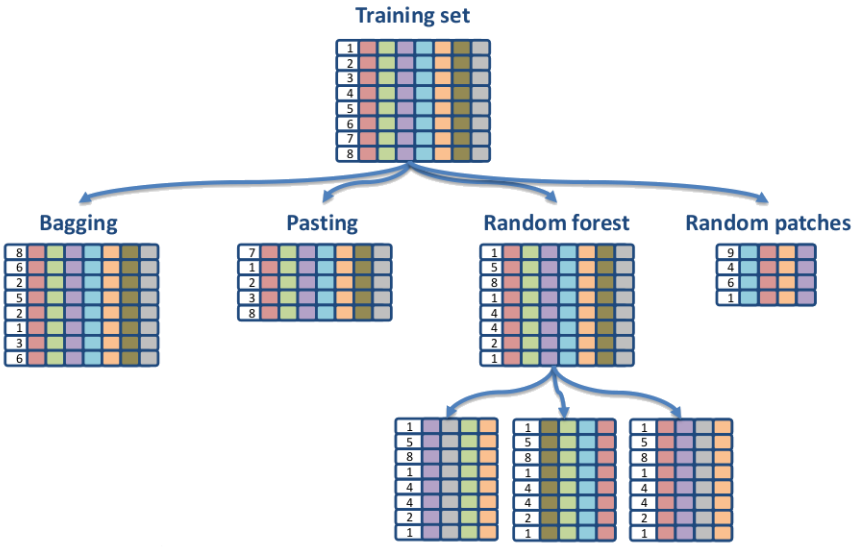


Figure 9.2: Visual representation of the random inducers algorithms.

dom subset of features and then determines the best split only over these features. In the random patches algorithm [Louppe and Geurts, 2012], base classifiers are created by randomly drawn bootstrap subsets of both examples and features. To further clarify the difference between the random inducer methods, in Figure 9.2, we show a visual representation of the random inducers algorithms.

9.2.2 *Combination methods*

Lastly, the base classifiers are combined using either majority voting, cost-sensitive weighted voting and cost-sensitive stacking. Majority voting consists in collecting the predictions of each base classifier and selecting the decision with the highest number of votes, see (9.1).

9.2.2.1 *Cost-sensitive weighted voting*

This method is an extension of weighted voting. First, in the traditional approach, a similar comparison of the votes of the base classifiers is made, but giving a weight α_j to each classifier M_j during the voting phase [Zhou, 2012]

$$f_{wv}(\mathcal{S}, \mathcal{M}, \alpha) = \arg\max_{c \in \{0,1\}} \sum_{j=1}^{T} \alpha_j \mathbf{1}_c(M_j(\mathcal{S})), \tag{9.5}$$

where $\alpha = \{\alpha_j\}_{j=1}^{T}$. The calculation of α_j is related to the performance of each classifier M_j. It is usually defined as the normalized misclassification error ϵ of the base classifier M_j in the out of bag set $\mathcal{S}_j^{oob} = \mathcal{S} - \mathcal{S}_j$

$$\alpha_j = \frac{1 - \epsilon(M_j(\mathcal{S}_j^{oob}))}{\sum_{j_1=1}^{T} 1 - \epsilon(M_{j_1}(\mathcal{S}_{j_1}^{oob}))}. \tag{9.6}$$

However, as previously discussed, the misclassification measure is not suitable in many real-world classification problems. We herein propose a method to calculate the weights α_j taking into account the actual savings of the classifiers. Therefore using (3.7), we define

$$\alpha_j = \frac{Savings(M_j(\mathcal{S}_j^{oob}))}{\sum_{j_1=1}^{T} Savings(M_{j_1}(\mathcal{S}_j^{oob}))}. \tag{9.7}$$

This method guaranties that the base classifiers that contribute to a higher increase in savings have more importance in the ensemble.

9.2.2.2 *Cost-sensitive stacking*

The staking method consists in combining the different base classifiers by learning a second level algorithm on top of them [Wolpert, 1992]. In this framework, once the base classifiers are constructed using the training set $\mathcal{S}$, a new set is constructed where the output of the base classifiers are now considered as the features while keeping the class labels.

Even though there is no restriction on which algorithm can be used as a second level learner, it is common to use a linear model [Zhou, 2012], such as

$$f_s(\mathcal{S}, \mathcal{M}, \beta) = g\left(\sum_{j=1}^{T} \beta_j M_j(\mathcal{S})\right), \tag{9.8}$$

where $\beta = \{\beta_j\}_{j=1}^{T}$, and $g(\cdot)$ is the sign function $g(z) = sign(z)$ in the case of a linear regression or the sigmoid function, defined as $g(z) = 1/(1 + e^{-z})$, in the case of a logistic regression.

Moreover, following the logic used in [Nesbitt, 2010], we propose learning the set of parameters β using our proposed cost-sensitive logistic regression (CSLR) [Correa Bahnsen et al., 2014a]. The CSLR algorithm consists in introducing example-dependent costs into a logistic regression, by changing the objective function of the model to one that is cost-sensitive. For

the specific case of cost-sensitive stacking, we define the cost function as:

$$J(\mathcal{S}, \mathcal{M}, \beta) = \sum_{i=1}^{N} \left[y_i \left(f_s(\mathbf{x}_i, \mathcal{M}, \beta) \cdot (C_{TP_i} - C_{FN_i}) + C_{FN_i} \right) + (1 - y_i) \left(f_s(\mathbf{x}_i, \mathcal{M}, \beta) \cdot (C_{FP_i} - C_{TN_i}) + C_{TN_i} \right) \right]. \quad (9.9)$$

Then, the parameters β that minimize the logistic cost function are used in order to combine the different base classifiers. However, as discussed in [Correa Bahnsen et al., 2014a], this cost function is not convex for all possible cost matrices, therefore, we use genetic algorithms to minimize it.

Similarly to cost-sensitive weighting, this method guarantees that the base classifiers that contribute to a higher increase in savings have more importance in the ensemble. Furthermore, by learning an additional second level cost-sensitive method, the combination is made such that the overall savings measure is maximized.

9.2.3 *Algorithms*

Finally, Algorithm 9.1 summarizes the ECSDT methods. In total, we evaluate 12 different algorithms, as four different random inducers (bagging, pasting, random forest and random patches) and three different combinators (majority voting, cost-sensitive weighted voting and cost-sensitive stacking) can be selected in order to construct the cost-sensitive ensemble.

Algorithm 9.1. *The proposed* ECSDT *algorithms.*

Require: CSDT (an example-dependent cost-sensitive decision tree algorithm), T the number of iterations, $\mathcal{S}$ the training set, $inducer$, N_e number of examples for each base classifier, N_f number of examples for each base classifier, $combinator$.

1: **function** ECSDT($\mathcal{S}, T, inducer, N_e, N_f, combinator$)
2: **Step 1:** Create the set of base classifiers
3: **for** $j \leftarrow 1$ to T **do**
4: **switch**($inducer$)
5: **case** Bagging:
6: $\mathcal{S}_j \leftarrow$ Sample N_e examples from $\mathcal{S}$ with replacement.
7: **case** Pasting:
8: $\mathcal{S}_j \leftarrow$ Sample N_e examples from $\mathcal{S}$ without replacement.

case Random forests:
$\mathcal{S}_j \leftarrow$ Sample N_e examples from $\mathcal{S}$ with replacement.
case Random patches:
$\mathcal{S}_j \leftarrow$ Sample N_e examples and N_f features from $\mathcal{S}$ with replacement.
end switch
$M_j \leftarrow \text{CSDT}(\mathcal{S}_j)$
$\mathcal{S}_j^{oob} \leftarrow \mathcal{S} - \mathcal{S}_j$
$\alpha_j \leftarrow Savings(M_j(\mathcal{S}_j^{oob}))$
end for
Step 2: Combine the different base classifiers
switch(combinator)
case Majority voting:
$H \leftarrow f_{mv}(\mathcal{S}, \mathcal{M})$
case Cost-sensitive weighted voting:
$H \leftarrow f_{wv}(\mathcal{S}, \mathcal{M}, \alpha)$
case Cost-sensitive stacking:
$\beta \leftarrow \arg\min_{\beta \in \mathbb{R}^T} J(\mathcal{S}, \mathcal{M}, \beta)$
$H \leftarrow f_s(\mathcal{S}, \mathcal{M}, \beta)$
end switch
return H
end function

9.3 THEORETICAL ANALYSIS OF THE ECSDT

Although the above proposed algorithm is simple, there is no work that has formally investigated ensemble performance in terms other than accuracy. In this section, our aim is to prove theoretically that combining individual cost-sensitive classifiers achieves better results in the sense of higher savings.

We denote $\mathcal{S}_a$ $a \in \{0, 1\}$, as the subset of $\mathcal{S}$ where the examples belong to the class a:

$$\mathcal{S}_a = \{\mathbf{x}_i^* | y_i = a, i \in 1, \ldots, N\}, \tag{9.10}$$

where $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1$, $\mathcal{S}_0 \cap \mathcal{S}_1 = \varnothing$, and $N_a = |\mathcal{S}_a|$. Also, we define the average cost of the base classifiers as:

$$\overline{\text{Cost}}(\mathcal{M}(\mathcal{S})) = \frac{1}{T} \sum_{j=1}^{T} \text{Cost}(M_j(\mathcal{S})). \tag{9.11}$$

Firstly, we prove the following lemma that states the cost of an ensemble H on the subset $\mathcal{S}_a$ is lower than the average cost of the base classifiers on the same set for $a \in \{0, 1\}$.

Lemma 9.1. *Let* H *be an ensemble of* $T \geqslant 3$ *classifiers* $\mathcal{M} = \{M_1, M_2, \ldots, M_T\}$*, and* $\mathcal{S}$ *a testing set of size* N*. If each one of the base classifiers has a probability of being correct higher or equal than one half,* $p \geqslant \frac{1}{2}$*, and the reasonableness conditions of the cost matrix are satisfied, then the following holds true*

$$\text{Cost}(H(\mathcal{S}_a)) \leqslant \overline{\text{Cost}}(\mathcal{M}(\mathcal{S}_a)), \quad a \in \{0, 1\}, \tag{9.12}$$

Proof. First, we decompose the total cost of the ensemble by applying equations (3.2) and (3.3). Additionally, we separate the analysis for $a = 0$ and $a = 1$:

- $a = 0$:

$$\begin{aligned}\text{Cost}(H(\mathcal{S}_0)) = \sum_{i=1}^{N_0} & y_i(c_i C_{TP_i} + (1 - c_i)C_{FN_i}) + \\ & (1 - y_i)(c_i C_{FP_i} + (1 - c_i)C_{TN_i}).\end{aligned} \tag{9.13}$$

Moreover, we know from (9.2) that the probability of an ensemble making the right decision, i.e., $y_i = c_i$, for any given example, is equal to P_c. Therefore, we can use this probability to estimate the expected savings of an ensemble:

$$\text{Cost}(H(\mathcal{S}_0)) = \sum_{i=1}^{N_0} P_c C_{TN_i} + (1 - P_c)C_{FP_i}. \tag{9.14}$$

- $a = 1$:

In the case of $\mathcal{S}_1$, and following the same logic as when $a = 0$, the cost of an ensemble is:

$$\text{Cost}(H(\mathcal{S}_1)) = \sum_{i=1}^{N_1} P_c C_{TP_i} + (1 - P_c)C_{FN_i}. \tag{9.15}$$

The second part of the proof consists in analyzing the right hand side of (9.12), specifically, the average cost of the base classifiers on set $\mathcal{S}_a$. To do that, with the help of (3.2) and (9.11), we may express the average cost of the base classifiers as:

$$\overline{\text{Cost}}(\mathcal{M}(\mathcal{S}_a)) = \frac{1}{T}\sum_{j=1}^{T}\sum_{i=1}^{N_a} \text{Cost}(M_j(\mathbf{x}_i^*)). \tag{9.16}$$

We define the set of base classifiers that make a negative prediction as

$$\mathcal{T}_{i0} = \{M_j(\mathbf{x}_i^*) | M_j(\mathbf{x}_i^*) = 0, j \in 1, \ldots, T\}, \tag{9.17}$$

similarly, the set of classifiers that make a positive prediction as

$$\mathcal{T}_{i1} = \{M_j(\mathbf{x}_i^*) | M_j(\mathbf{x}_i^*) = 1, j \in 1, \ldots, T\}. \tag{9.18}$$

Then, by taking the cost of negative and positive predictions from (3.6), the average cost of the base learners becomes:

$$\begin{aligned}\overline{Cost}(\mathcal{M}(\mathcal{S}_a)) = &\frac{1}{T}\sum_{i=1}^{N_a}\Big(|\mathcal{T}_{i0}| \cdot Cost(f_0(\mathbf{x}_i^*)) \\ &+ |\mathcal{T}_{i1}| \cdot Cost(f_1(\mathbf{x}_i^*))\Big).\end{aligned} \tag{9.19}$$

We separate the analysis for $a = 0$ and $a = 1$:

- $a = 0$:

$$\overline{Cost}(\mathcal{M}(\mathcal{S}_0)) = \sum_{i=1}^{N_0}\left(\frac{|\mathcal{T}_{i0}|}{T} \cdot C_{TN_i} + \frac{|\mathcal{T}_{i1}|}{T} \cdot C_{FP_i}\right). \tag{9.20}$$

Furthermore, we know from (9.2) that an average base classifier will have a correct classification probability of ρ, then $\frac{|\mathcal{T}_{i0}|}{T} = \rho$, leading to:

$$\overline{Cost}(\mathcal{M}(\mathcal{S}_0)) = \sum_{i=1}^{N_0}\rho \cdot C_{TN_i} + (1-\rho) \cdot C_{FP_i}. \tag{9.21}$$

- $a = 1$:

Similarly, for the set $\mathcal{S}_1$, the average classifier will have a correct classification probability of ρ, then $\frac{|\mathcal{T}_{i1}|}{T} = \rho$.
Therefore,

$$\overline{Cost}(\mathcal{M}(\mathcal{S}_1)) = \sum_{i=1}^{N_1}\rho \cdot C_{TP_i} + (1-\rho) \cdot C_{FN_i}. \tag{9.22}$$

Finally, by replacing in (9.12) the expected savings of an ensemble with (9.14) for $a = 0$ and (9.15) for $a = 1$, and the average cost of the base learners with (9.21) for $a = 0$ and (9.22) for $a = 1$, (9.12) is rewritten as:
for $a = 0$:

$$\sum_{i=1}^{N_0} P_c C_{TN_i} + (1-P_c)C_{FP_i} \leqslant \sum_{i=1}^{N_0}\rho \cdot C_{TN_i} + (1-\rho) \cdot C_{FP_i}, \tag{9.23}$$

for $a = 1$:

$$\sum_{i=1}^{N_1} P_c C_{TP_i} + (1 - P_c) C_{FN_i} \leqslant \sum_{i=1}^{N_1} \rho \cdot C_{TP_i} + (1 - \rho) \cdot C_{FN_i}. \tag{9.24}$$

Since $\rho \geqslant \frac{1}{2}$, then $P_c \geqslant \rho$ from (9.4), and using the *reasonableness conditions* described in Section 3.3, i.e, $C_{FP_i} > C_{TN_i}$ and $C_{FN_i} > C_{TP_i}$, we find that (9.23) and (9.24) are True. □

Lemma 9.1 separates the costs on sets S_0 and S_1. We are interested in analyzing the overall savings of an ensemble. In this direction, we demonstrate in the following theorem, that the expected savings of an ensemble of classifiers are higher than the expected average savings of the base learners.

Theorem 9.2. *Let* H *be an ensemble of* $T \geqslant 3$ *classifiers* $\mathcal{M} = \{M_1, \ldots, M_T\}$, *and* S *a testing set of size* N, *then the expected savings of using* H *in* S *are lower than the average expected savings of the base classifiers, in other words,*

$$\mathrm{Savings}(H(S)) \geqslant \overline{\mathrm{Savings}}(\mathcal{M}(S)). \tag{9.25}$$

Proof. Given (3.7), (9.25) is equivalent to

$$\mathrm{Cost}(H(S)) \leqslant \overline{\mathrm{Cost}}(\mathcal{M}(S)). \tag{9.26}$$

Afterwards, by applying the cost definition of (3.3), and grouping the sets of negative and positive examples using (9.10), (9.26) becomes

$$\sum_{a \in \{0,1\}} \mathrm{Cost}(H(S_a)) \leqslant \sum_{a \in \{0,1\}} \overline{\mathrm{Cost}}(\mathcal{M}(S_a)), \tag{9.27}$$

which can be easily proved using Lemma 9.1, since, if the cost of an ensemble H is lower than the average cost of the base classifiers on both S_0 and S_1, implies that it is also lower on the sum of the cost on both sets, therefore, proving Theorem 9.2. □

9.4 EXPERIMENTS

For the experiments we used five datasets from four different real world example-dependent cost-sensitive problems: Credit card fraud detection (see Section 4.1), credit scoring (see Section 4.2), churn modeling (see Section 5.1) and direct marketing (see Section 5.2). The different datasets are summarized in Table 6.1.

We first used three classification algorithms, decision tree (DT), logistic regression (LR) and random forest (RF). Using the implementation of *Scikit-learn* [Pedregosa et al., 2011], each algorithm is trained using the different training sets: training (t), under-sampling (u), cost-proportionate rejection-sampling (r) [Zadrozny et al., 2003] and cost-proportionate over-sampling (o) [Elkan, 2001]. Afterwards, we evaluate the results of the algorithms using BMR [Correa Bahnsen et al., 2014b]. Then, the cost-sensitive logistic regression (CSLR) [Correa Bahnsen et al., 2014a] and cost-sensitive decision tree (CSDT) [Correa Bahnsen et al., 2015a] were also evaluated. Lastly, we calculate the proposed ensembles of cost-sensitive decision trees algorithms. In particular, using each of the random inducer methods, bagging (CSB), pasting (CSP), random forests (CSRF) and random patches (CSRP), and then blending the base classifiers using each one of the combination methods; majority voting (mv), cost-sensitive weighted voting (*wv*) and cost-sensitive stacking (s). Unless otherwise stated, the random selection of the training set was repeated 50 times, and in each time the models were trained and results collected, this allows us to measure the stability of the results. The implementation of the cost-sensitive algorithms is done using the *CostCla* library, see Appendix A.

The results are shown in Table 9.1. First, when observing the results of the cost-insensitive methods (CI), that is, DT, LR and RF algorithms trained on the t and u sets, the RF algorithm produces the best result by savings in three out of the five sets, followed by the LR $-$ u. It is also clear that the results on the t dataset are not as good as the ones on the u, this is highly related to the unbalanced distribution of the positives and negatives in all the databases.

In the case of cost-proportionate sampling methods (CPS), specifically the cost-proportionate rejection sampling (r) and cost-proportionate over sampling (o). It is observed than in four cases the savings increases quite significantly. It is on the fraud detection database where these methods do not outperform the

Family	Algorithm	Fraud	Churn	Credit 1
CI	DT-t	0.3176±0.0357	-0.0018±0.0194	0.1931±0.0087
	LR-t	0.0092±0.0002	-0.0001±0.0002	0.0177±0.0126
	RF-t	0.3342±0.0156	-0.0026±0.0079	0.1471±0.0071
	DT-u	0.5239±0.0118	-0.0389±0.0583	0.3287±0.0125
	LR-u	0.1243±0.0387	0.0039±0.0492	0.4118±0.0313
	RF-u	0.5684±0.0097	0.0433±0.0533	0.4981±0.0079
CPS	DT-r	0.3439±0.0453	0.0054±0.0568	0.3310±0.0126
	LR-r	0.3077±0.0301	0.0484±0.0375	0.3965±0.0263
	RF-r	0.3812±0.0264	0.1056±0.0412	0.4989±0.0080
	DT-o	0.3172±0.0274	0.0251±0.0195	0.1738±0.0092
	LR-o	0.2793±0.0185	0.0316±0.0228	0.3301±0.0109
	RF-o	0.3612±0.0295	0.0205±0.0156	0.2128±0.0081
BMR	DT-t-BMR	0.6045±0.0386	0.0298±0.0145	0.1054±0.0358
	LR-t-BMR	0.4552±0.0203	0.1082±0.0316	0.2189±0.0541
	RF-t-BMR	0.6414±0.0154	0.0856±0.0354	0.4924±0.0087
CST	CSLR-t	0.6113±0.0262	0.1118±0.0484	0.4554±0.1039
	CSDT-t	0.7116±0.2557	0.1115±0.0378	0.4829±0.0098
ECSDT	CSB-mv-t	0.7124±0.0162	0.1237±0.0368	0.4862±0.0102
	CSB-wv-t	0.7276±0.0116	0.1539±0.0255	0.4862±0.0102
	CSB-s-t	0.7181±0.0109	0.1441±0.0364	0.4847±0.0096
	CSP-mv-t	0.7106±0.0113	0.1227±0.0399	0.4853±0.0104
	CSP-wv-t	0.7244±0.0202	0.1501±0.0302	0.4854±0.0105
	CSP-s-t	0.7212±0.0067	0.1488±0.0272	0.4848±0.0084
	CSRF-mv-t	0.6498±0.0598	0.0300±0.0488	0.4980±0.0120
	CSRF-wv-t	0.7249±0.0742	0.0624±0.0477	0.4979±0.0124
	CSRF-s-t	0.6731±0.0931	0.0586±0.0507	0.4839±0.0160
	CSRP-mv-t	0.7220±0.0082	0.1321±0.0280	**0.5154±0.0077**
	CSRP-wv-t	**0.7348±0.0131**	0.1615±0.0252	0.5152±0.0083
	CSRP-s-t	0.7336±0.0108	**0.1652±0.0264**	0.4989±0.0088

(those models with the highest savings are market as bold)

Table 9.1: Results of the algorithms measured by savings

algorithms trained on the under-sampled set. This may be related to the fact that in this database the initial percentage of positives is 1.5% which is similar to the percentage in the r and o sets. However it is 50.42% in the u set, which may help explain why this method performs much better as measured by savings.

Afterwards, in the case of the BMR algorithms, the results show that this method outperforms the previous ones in four

Family	Algorithm	Credit 2	Marketing
CI	DT-t	-0.0616±0.0229	-0.2342±0.0609
	LR-t	0.0039±0.0012	-0.2931±0.0602
	RF-t	0.0303±0.0040	-0.2569±0.0637
	DT-u	-0.1893±0.0314	-0.0278±0.0475
	LR-u	0.1850±0.0231	0.2200±0.0376
	RF-u	0.1237±0.0228	0.1227±0.0443
CPS	DT-r	0.0724±0.0212	0.1960±0.0527
	LR-r	0.2650±0.0115	0.4210±0.0267
	RF-r	0.3055±0.0106	0.3840±0.0360
	DT-o	0.0918±0.0225	-0.2598±0.0559
	LR-o	0.2554±0.0090	0.3129±0.0277
	RF-o	0.2242±0.0070	-0.1782±0.0618
BMR	DT-t-BMR	0.2740±0.0067	0.4598±0.0089
	LR-t-BMR	**0.3148±0.0094**	**0.4973±0.0084**
	RF-t-BMR	0.3133±0.0094	0.4807±0.0093
CST	CSLR-t	0.2748±0.0069	0.4484±0.0072
	CSDT-t	0.2835±0.0078	0.4741±0.0063
ECSDT	CSB-mv-t	0.2945±0.0105	0.4837±0.0078
	CSB-wv-t	0.2948±0.0106	0.4838±0.0079
	CSB-s-t	0.2856±0.0088	0.4769±0.0078
	CSP-mv-t	0.2919±0.0097	0.4831±0.0081
	CSP-wv-t	0.2921±0.0098	0.4832±0.0082
	CSP-s-t	0.2870±0.0084	0.4752±0.0089
	CSRF-mv-t	0.2274±0.0520	0.3929±0.0655
	CSRF-wv-t	0.2948±0.0079	0.4728±0.0125
	CSRF-s-t	0.2518±0.0281	0.3854±0.0899
	CSRP-mv-t	0.3053±0.0087	0.4960±0.0075
	CSRP-wv-t	0.3015±0.0086	0.4885±0.0076
	CSRP-s-t	0.2956±0.0078	0.4878±0.0080

(those models with the highest savings are market as bold)

Table 9.2: Continuation of Table 9.1

cases and has almost the same result in the other set. In the fraud detection set, the results are quite better, since the savings of the three classification algorithms increase when using this methodology. The next family of algorithms is the cost-sensitive training, which includes the CSLR and CSDT techniques. In this case, only in two databases the results are improved. Lastly, we evaluate the proposed ECSDT algorithms. The results show

Family	Algorithm	F-Rank	perBest
ECSDT	CSRP-wv-t	2.6	98.35
ECSDT	CSRP-s-t	3.4	97.72
ECSDT	CSRP-mv-t	4.0	94.99
ECSDT	CSB-wv-t	5.6	95.49
ECSDT	CSP-wv-t	7.4	94.72
ECSDT	CSB-mv-t	8.2	91.39
ECSDT	CSRF-wv-t	9.4	84.35
BMR	RF-t-BMR	9.4	86.16
ECSDT	CSP-s-t	9.6	93.80
ECSDT	CSP-mv-t	10.2	91.00
ECSDT	CSB-s-t	10.2	93.12
BMR	LR-t-BMR	11.2	73.98
CPS	RF-r	11.6	77.37
CST	CSDT-t	12.6	88.69
CST	CSLR-t	14.4	83.34
ECSDT	CSRF-mv-t	15.2	70.88
ECSDT	CSRF-s-t	16.0	75.68
CI	RF-u	17.2	52.83
CPS	LR-r	19.0	63.39
BMR	DT-t-BMR	19.0	60.05
CPS	LR-o	21.0	53.05
CPS	DT-r	22.6	35.33
CI	LR-u	22.8	40.43
CPS	RF-o	22.8	34.81
CI	DT-u	24.4	27.01
CPS	DT-o	25.0	24.25
CI	DT-t	26.0	16.14
CI	RF-t	26.2	16.73
CI	LR-t	28.0	1.19

Table 9.3: Savings Friedman ranking and average percentage of best result

that these methods arise to the best overall results in three sets, while being quite competitive in the others.

Subsequently, in order to statistically sort the classifiers we computed the Friedman ranking (F-Rank) statistic [Demšar, 2006]. This rank increases with the cost of the algorithms. We also calculate the average savings of each algorithm compared with the highest savings in each set (perBest), as a measure of how close are the savings of an algorithm to the best result.

Algorithm	Fraud	Churn	Credit1	Credit2	Marketing
RF-u	17	18	5	23	23
RF-r	20	13	3	3	19
RF-t-BMR	14	14	8	2	9
CSDT-t	10	11	16	14	12
CSRP-wv-t	1	2	2	5	3

Table 9.4: Savings ranks of best algorithm of each family by database

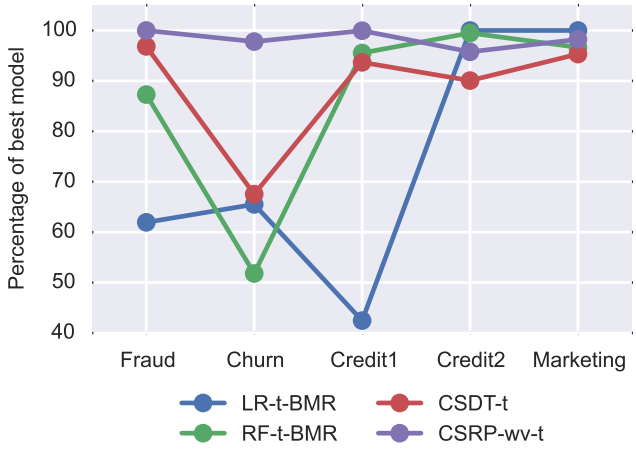
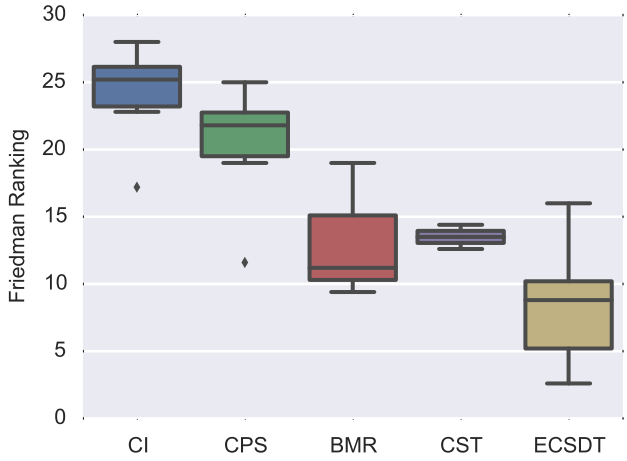


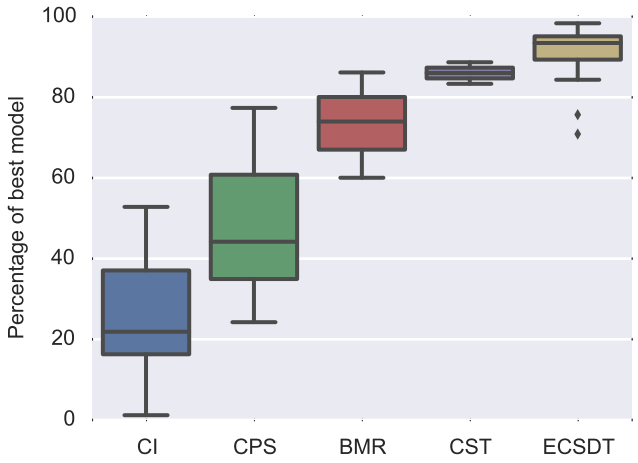
Figure 9.3: **Comparison of the savings of the algorithms versus the highest savings in each database.** The CSRP − *w*t is very close to the best result in all the databases. Additionally, even though the LR − BMR is the best algorithm in two databases, the performance in the other three is very poor.

In Table 9.3, the results are shown. It is observed that the first six algorithms, according to the F-Rank, belong to the ECSDT family. In particular, the best three classifiers is the ensemble of cost-sensitive decision trees using the random patches approach. Giving the best result the one that blends the base classifiers using weighted voting method. Moreover as shown in Table 9.4, this method ranks on each dataset 1^{st}, 2^{nd}, 2^{nd}, 5^{th} and 3^{rd}, respectively. For comparison the best method from an other family is the RF with BMR, which ranks 14^{th}, 14^{th}, 8^{th}, 2^{nd} and 9^{th}.

Moreover, when analyzing the perBest statistic, it is observed that it follows almost the same order as the F-Rank. Notwithstanding, there are cases in which algorithms ranks are different in the two statistics, for example the CSDT − t algorithm has a lower F-Rank than the RF − BMR, but the perBest if better. This happens because, the F-Rank does not take into account



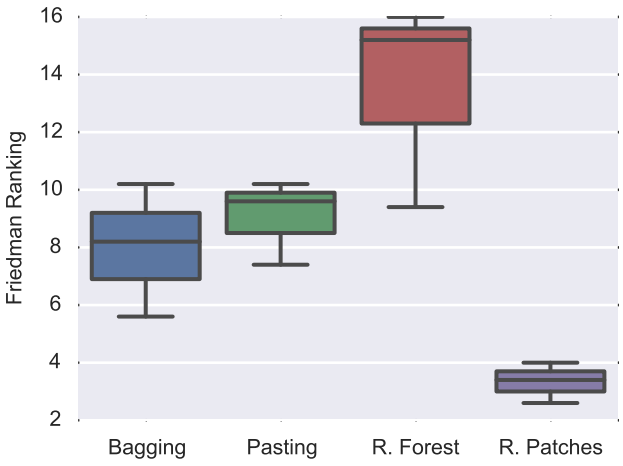
(a) Comparison of the Friedman ranking.



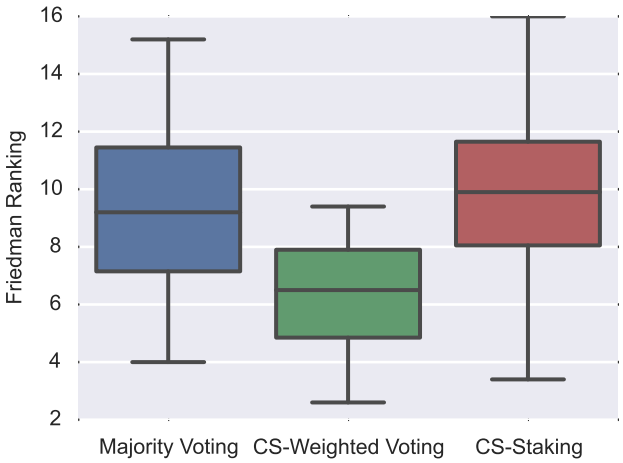
(b) Comparison of the average savings of the algorithms versus the highest savings.

Figure 9.4: **Comparison of the results by family of classifiers.** The ECSDT family has the best performance measured either by Friedman ranking or average percentage of best model.

the difference in savings within algorithms. This can be further investigated in Figure 9.3. Even that ranks of the BMR models are better than the CSDT, the latter is on average closer to the best performance method in each set. Moreover, it is confirmed that the CSRP $-$ *wt* is very close to the best result in all cases. Lastly, it is shown why the F-Rank of the LR $-$ BMR is high, given the fact that is the best model in two databases. The reason for that, is because the performance on the other sets is very poor.



(a) Comparison by inducer methodology.



(b) Comparison by combination of base classifiers approach.

Figure 9.5: **Comparison of the Friedman ranking within the** ECSDT **family.** Overall, the random inducer method that provides the best results is the CSRP. Moreover, the best combination method compared by Friedman ranking is the cost-sensitive weighted voting.

Furthermore Figure 9.4a, shows the Friedman ranking of each family of classifiers. The ECSDT methods are overall better, followed by the BMR and the CST methods. As expected, the CI family is the one that performs the worst. Nevertheless, there is a significant variance within the ranks in the ECSDT family, as the best one has a Friedman ranking of 2.6 and the worst 16. Similar results are found when observing the perBest shown in Figure 9.4b. However, in the case of the perBest, the CST methods perform better than the BMR. It is important, in

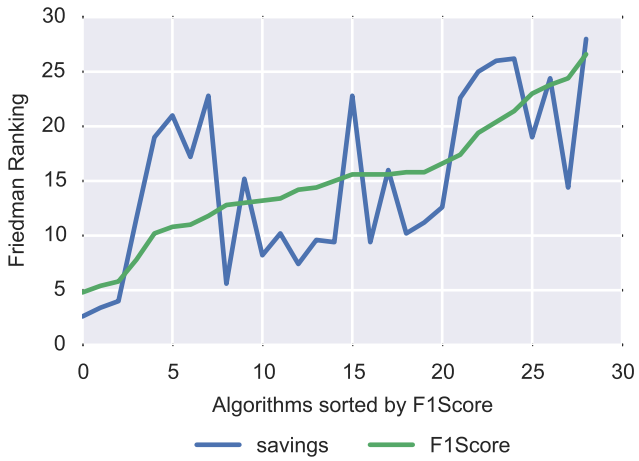


Figure 9.6: **Comparison of the Friedman ranking of the savings and F1Score sorted by F1Score ranking.** The best two algorithms according to their Friedman rank of F1Score are indeed the best ones measured by the Friedman rank of the savings. However, this relation does not consistently hold for the other algorithms as the correlation between the rankings is just 65.10%.

both cases it is confirmed that the ECSDT family of methods is the one that arise to the best results as measured by savings.

We further investigate the different methods that compose the ECSDT family, first by inducer methods and by the combination approach. In Figure 9.5a, the Friedman ranking of the ECSDT methods grouped by inducer algorithm are shown. It is observed that the worst method is the random forest methodology. This may be related to the fact that within the random inducer methods, this is the only one that also modified the learner algorithm in the sense that it randomly select features for each step during the decision tree growing. Moreover, as expected the bagging and pasting methods perform quite similar, after all the only difference is that in bagging the sampling is done with replacement, while it is not the case in pasting. In general the best methodology is random patches. Additionally, in Figure 9.5b, a similar analysis is made taking into account the combination of base classifiers approach. In this case, the best combination method is weighted voting, while majority voting and staking have a similar performance.

Finally, in Table 9.5 the results of the algorithms measured by F1Score are shown. It is observed that the model with the highest savings is not the same as the one with the highest

Family	Algorithm	Fraud	Churn	Credit 1
CI	DT-t	0.4458±0.0133	0.0733±0.0198	0.2593±0.0068
	LR-t	0.1531±0.0045	0.0000±0.0000	0.0494±0.0277
	RF-t	0.2061±0.0041	0.0249±0.0146	0.2668±0.0085
	DT-u	0.1502±0.0066	0.1175±0.0103	0.2276±0.0044
	LR-u	0.0241±0.0163	0.1222±0.0098	0.3160±0.0314
	RF-u	0.0359±0.0065	0.1346±0.0112	0.3193±0.0053
CPS	DT-r	0.4321±0.0086	0.1206±0.0132	0.2310±0.0049
	LR-r	0.1846±0.0123	0.1258±0.0111	0.3597±0.0156
	RF-r	0.2171±0.0100	0.1450±0.0131	0.3361±0.0067
	DT-o	**0.4495±0.0063**	0.1022±0.0180	0.2459±0.0081
	LR-o	0.1776±0.0117	0.1085±0.0203	0.3769±0.0067
	RF-o	0.2129±0.0080	0.0841±0.0201	0.3281±0.0078
BMR	DT-t-BMR	0.2139±0.0215	0.0941±0.0157	0.1514±0.0390
	LR-t-BMR	0.1384±0.0044	0.1370±0.0150	0.1915±0.0340
	RF-t-BMR	0.2052±0.0183	0.1264±0.0156	0.3186±0.0072
CST	CSLR-t	0.2031±0.0065	0.1134±0.0151	0.1454±0.0517
	CSDT-t	0.2522±0.0980	0.1288±0.0194	0.2754±0.0059
ECSDT	CSB-mv-t	0.2112±0.0125	0.1481±0.0122	0.2927±0.0108
	CSB-wv-t	0.2112±0.0091	0.1686±0.0125	0.2926±0.0108
	CSB-s-t	0.2072±0.0103	0.1554±0.0121	0.2818±0.0075
	CSP-mv-t	0.2098±0.0126	0.1480±0.0136	0.2903±0.0108
	CSP-wv-t	0.2099±0.0054	0.1651±0.0120	0.2905±0.0110
	CSP-s-t	0.2064±0.0069	0.1590±0.0099	0.2809±0.0062
	CSRF-mv-t	0.2208±0.0022	0.1081±0.0224	0.2994±0.0226
	CSRF-wv-t	0.2175±0.0019	0.1220±0.0234	0.2992±0.0236
	CSRF-s-t	0.2169±0.0045	0.1304±0.0162	0.2916±0.0236
	CSRP-mv-t	0.2691±0.0054	0.1511±0.0110	0.4031±0.0079
	CSRP-wv-t	0.2780±0.0041	**0.1710±0.0140**	**0.4049±0.0066**
	CSRP-s-t	0.2735±0.0148	0.1622±0.0103	0.3953±0.0141

(those models with the highest F1Score are market as bold)

Table 9.5: Results as measured by F1Score

F1Score in all of the databases, corroborating the conclusions from [Correa Bahnsen et al., 2013], as selecting a method by a traditional statistic does not give the same result as selecting it using a business oriented measure such as financial savings. This can be further examined in Figure 9.6, where the ranking of the F1Score and savings are compared. It is observed that the best two algorithms according to their Friedman rank of F1Score are indeed the best ones measured by the Friedman

Family	Algorithm	Credit 2	Marketing
CI	DT-t	0.2614±0.0083	0.2647±0.0079
	LR-t	0.0155±0.0037	0.2702±0.0125
	RF-t	0.0887±0.0061	0.2884±0.0116
	DT-u	0.3235±0.0055	0.2659±0.0061
	LR-u	**0.3890±0.0053**	0.3440±0.0083
	RF-u	0.3815±0.0051	0.3088±0.0065
CPS	DT-r	0.3409±0.0046	0.2739±0.0076
	LR-r	0.3793±0.0049	0.3374±0.0101
	RF-r	0.3570±0.0048	0.3103±0.0075
	DT-o	0.3258±0.0056	0.2634±0.0089
	LR-o	0.3804±0.0044	**0.3568±0.0102**
	RF-o	0.3212±0.0054	0.3083±0.0093
BMR	DT-t-BMR	0.3338±0.0052	0.2433±0.0071
	LR-t-BMR	0.3572±0.0045	0.2954±0.0079
	RF-t-BMR	0.3551±0.0053	0.2744±0.0070
CST	CSLR-t	0.3363±0.0045	0.2339±0.0051
	CSDT-t	0.3483±0.0046	0.2680±0.0060
ECSDT	CSB-mv-t	0.3503±0.0046	0.2758±0.0072
	CSB-wv-t	0.3503±0.0046	0.2757±0.0072
	CSB-s-t	0.3533±0.0046	0.2799±0.0074
	CSP-mv-t	0.3499±0.0044	0.2749±0.0064
	CSP-wv-t	0.3498±0.0045	0.2749±0.0064
	CSP-s-t	0.3524±0.0046	0.2778±0.0104
	CSRF-mv-t	0.3560±0.0118	0.2780±0.0106
	CSRF-wv-t	0.3549±0.0069	0.2552±0.0187
	CSRF-s-t	0.3540±0.0071	0.2578±0.0186
	CSRP-mv-t	0.3743±0.0050	0.2916±0.0062
	CSRP-wv-t	0.3721±0.0051	0.2781±0.0063
	CSRP-s-t	0.3720±0.0049	0.2922±0.0137

(those models with the highest F1Score are market as bold)

Table 9.6: Continuation of Table 9.5

rank of the savings. However, this relation does not consistently hold for the other algorithms as the correlation between the rankings is just 65.10%.

10 CONCLUSIONS

This thesis deals with the problem of example-dependent cost-sensitive classification. Several real-world business applications of classification models are example-dependent cost-sensitive, in the sense that the objective of using an algorithm is related to maximizing the profit of the company. Moreover, the different costs due to misclassification vary among examples. Particularly, we focused on four different real-world applications: credit card fraud detection, credit scoring, churn modeling and direct marketing. In all cases, evaluating a classification algorithm using traditional statistics such as misclassification rate or F_1Score, do not accurately represent the business oriented goals. Moreover, we founded, significant differences in the results, when using tradition cost-insensitive methods against example-dependent cost-sensitive methods.

First in Part I, we laid out the background for general classification methods, and their respective evaluation measures. Moreover, we introduce the problem of cost-sensitive learning, in particular, we define the differences between class-dependent and example-dependent problems. In addition, we present a evaluation measure that takes into account the real financial gains and losses of practical applications.

Then in Part II, we analyzed and discussed four real-world classification problems, that are the focus of this thesis, in particular, credit card fraud detection, credit scoring, churn modeling and direct marketing. In general, we showed why each of the applications is example-dependent cost-sensitive, and we elaborated a framework for the analysis of each problem. The part is organized in two chapters.

Finally in Part III, we introduced our proposed example-dependent cost-sensitive methods. First, we presented our Bayes minimum risk method. This method worked by including comparing the expected financial losses of different outcomes when classifying the examples in the different classes. Then, we focused on introducing the costs to the algorithms during the training phase. In particular, we presented the cost-sensitive logistic regression and cost-sensitive decision trees algorithms. The cost-sensitive decision tree algorithm proved to

be highly effective while maintaining the simplicity and interpretability of decision trees. However, this method suffer from high variance, to overcome that limitation, we proposed framework for ensembles of cost-sensitive decision trees. We have shown theoretically and experimentally that that the method ensembles of cost-sensitive decision trees ranks the best and outperforms state-of-the-art example-dependent cost-sensitive methodologies, when measured by financial savings. Lastly, in Appendix A, we presented the library *CostCla* that we developed as part of the thesis. This library is an open-source implementation of all the algorithms covered in this manuscript.

Finally, this thesis showed the importance of using the real example-dependent financial costs associated with real-world applications. In particular, we found significant differences in the results when evaluating a model using a traditional cost-insensitive measure such as the accuracy or F1Score, than when using the savings, leading to the conclusion of the importance of using the real practical financial costs of each context.

10.1 FUTURE RESEARCH DIRECTIONS

We foresee that the framework we developed though this thesis should open the door to developing more business focused algorithms, and that ultimately, the use of the actual financial costs during training will become a common practice. There are a list of points related to the work that can be further investigated. In the following, we address some issues.

- **Multi-class example-dependent cost-sensitive classification.** This thesis was focused on binary cost-sensitive classification problems. Nevertheless, we understand that not all the cost-sensitive applications are two-class problems. Some studies have start to look into Multi-class class-dependent cost-sensitive classification [Zhou and Liu, 2010]. Therefore, we expect that an interesting line of future work should include the expanding of our framework to multi-class problems.

- **Cost-sensitive calibration.** When evaluating how well a set of probabilities are calibrated, for example using the Brier score, the measure does not take into account the cost of each example. Therefore, the calibration methods that attempt to improve those kind of measures, as the ones presented in Section 6.2, also fail to take into account

the real cost-sensitive costs related to each application. Future work should include a deep analysis of the impact of the costs during the calibration of probabilities.

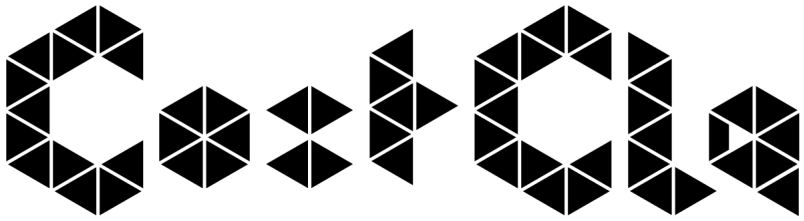
- **Staking cost-sensitive decision trees.** Even though we explore the method of staking in Section 9.2.2.2, we foreseen that a deeper analysis of the impact of having several layers of example-dependent cost-sensitive methods could enhance the performance of the system.

- **Example-dependent cost-sensitive boosting.** In Chapter 9, we only focused our attention to independent ensemble methods, in particular bagging algorithms. However, there is an other branch of ensemble methods, those that are dependent, such as boosting [Schapire, 1990], adaboost [Freund and Schapire, 1996] or gradient boosting [Friedman, 2001, 2002]. For some applications these methods have proved to outperform the bagging algorithms [Zhou, 2012], therefore, we think that future research should focus creating a framework for example-dependent cost-sensitive boosting algorithms.

- **Online example-dependent cost-sensitive classification.** The methods covered in this thesis are all batch, in the sense that the batch algorithms keeps the system weights constant while calculating the evaluation measures. However in some applications such as fraud detection, the evolving patters due to change in the fraudsters behavior is not capture by using batch methods. Therefore, the need for investigate this problem from an online-learning perspective [Pozzolo et al., 2014b]. In particular, we think that a first approach may consist in expanding the online class-dependent method proposed in [Wang et al., 2014], into an example-dependent cost-sensitive setting.

A

COSTCLA: A COST-SENSITIVE CLASSIFICATION LIBRARY IN **PYTHON**



CostCla is a **Python** open source cost-sensitive classification library built on top of *Scikit-learn*, *Pandas* and *Numpy*. *CostCla* provides a wide range of state-of-the-art example-dependent cost-sensitive methods for binary classification tasks. Source code, binaries and documentation are distributed under 3-Clause BSD license in the website `http://albahnsen.com/CostSensitiveClassification/`.

A.1 INTRODUCTION

Classification, in the context of machine learning, deals with the problem of predicting the class of a set of examples given their features. Traditionally, classification methods aim at minimizing the misclassification of examples, in which an example is misclassified if the predicted class is different from the true class. **Python** offers several machine learning packages, some including classification algorithms like *Scikit-learn* [Pedregosa et al., 2011], *PyBrain* [Schaul and Felder, 2010], *PyMC* [Patil et al., 2010], *mlpy* [Albanese et al., 2012] and *Orange* [Demšar et al., 2013].

Nevertheless, all this packages are based on a cost-insensitive framework, in which all misclassification errors carry the same cost. This is not the case in many real-world applications. For example in credit card fraud detection, failing to detect a fraudulent transaction may have an economical impact from a few to thousands of Euros, depending on the particular transaction and card holder [Ngai et al., 2011]. In churn modeling, a

model is used for predicting which customers are more likely to abandon a service provider. In this context, failing to identify a profitable or unprofitable churner has a significant different economic result [Verbraken et al., 2013]. Similarly, in direct marketing, wrongly predicting that a customer will not accept an offer when in fact he will, may have different financial impact, as not all customers generate the same profit [Zadrozny et al., 2003]. Lastly, in credit scoring, accepting loans from bad customers does not have the same economical loss, since customers have different credit lines, therefore, different profit [Verbraken et al., 2014].

In this chapter, we present *CostCla* a cost-sensitive classification library in **Python**. The library incorporates several cost-sensitive algorithms. Moreover, we show the huge differences in profit when using traditional machine learning algorithms versus cost-sensitive algorithms, on several real-world databases.

A.2 LIBRARY OVERVIEW

The core of the library consists of a number of state-of-the-art example-dependent cost-sensitive classification algorithms, such as cost-proportionate rejection-sampling [Zadrozny et al., 2003], cost-proportionate over-sampling [Elkan, 2001], Bayes minimum risk [Correa Bahnsen et al., 2013, 2014b], cost-sensitive logistic regression [Correa Bahnsen et al., 2014a], cost-sensitive decision trees [Correa Bahnsen et al., 2015a] and the cost-sensitive ensemble methods: cost-sensitive bagging, cost-sensitive pasting, cost-sensitive random forest and cost-sensitive random patches [Correa Bahnsen et al., 2015c].

Moreover, the library also includes three different example-dependent cost-sensitive databases. In particular two credit scoring databases, and one direct marketing database.

A.3 USAGE

In this section we provided a quick example of the usage of the *CostCla* library, and compare the results of different algorithms using a credit scoring database.

- Prepare dataset and load libraries

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
from costcla.datasets import load_creditscoring2
from costcla.sampling import cost_sampling
from costcla.metrics import savings_score
from costcla import models
data = load_creditscoring2()
X_train, X_test, y_train, y_test,
cost_mat_train, cost_mat_test = \
train_test_split(data.data, data.target, data.cost_mat)
```

- Random forest

```
f_RF = RandomForestClassifier()
f_RF.fit(X_train, y_train)
y_pred = f_RF.predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)
```

0.042197359989

- Cost-proportionate rejection sampling

```
X_cps_r, y_cps_r, cost_mat_cps_r =  \
cost_sampling(X_train, y_train, cost_mat_train,
              method='RejectionSampling')
f_RF.fit(X_train, y_train)
y_pred = f_RF.predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)
```

0.280743761779

- Bayes minimum risk

```
f_RF.fit(X_train, y_train)
y_prob_test = f_RF.predict_proba(X_test)
f_BMR = models.BayesMinimumRiskClassifier()
f_BMR.fit(y_test, y_prob_test)
y_pred = f_BMR.predict(y_prob_test, cost_mat_test)
print savings_score(y_test, y_pred, cost_mat_test)
```

0.285102564249

- Cost-sensitive decision tree

```
f_CSDT = models.CSDecisionTreeClassifier()
f_CSDT.fit(X_train, y_train, cost_mat_train)
y_pred = f_CSDT.predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)
```

0.289489571352

- Cost-sensitive random patches

```
f_CSRP = models.CSRandomPatchesClassifier()
f_CSRP.fit(X_train, y_train, cost_mat_train)
y_pred = f_CSRP.predict(X_test)
print savings_score(y_test, y_pred, cost_mat_test)
```

0.306607400467

In the Table A.1 we summarized the results of the different algorithms. It is observed that the library follows the same API structure of *Scikit-learn*, therefore, making *CostCla* highly compatible with the most widely used machine learning library in **Python**[1]. Moreover, the results highlight the importance of using example-dependent cost-sensitive methods when [TODO] complete, add time spend

Algorithm	Savings
Random forest	0.0422
Cost-proportionate rejection sampling	0.2807
Bayes minimum risk	0.2851
Cost-sensitive decision tree	0.2895
Cost-sensitive random patches	0.3066

Table A.1: Results of the different algorithms using the *CostCla* library

A.4 INSTALLATION

CostCla requires some prerequisite packages to be previously installed. Nevertheless, all are cross-platform an freely available online:

- **Python** version $\geqslant$ 2.7
- *Numpy* version $\geqslant$ 1.8.0
- *Pandas* version $\geqslant$ 0.14.0
- *Scikit-learn* version $\geqslant$ 0.15.0b2
- *pyea* version $\geqslant$ 0.1

The easiest way to install *CostCla* is with *pip*:

```
pip install costcla
```

1 http://machinelearningmastery.com/best-programming-language-for-machine-learning/

A.5 CONCLUSION

CostCla is a easy to use **Python** library for example-dependent cost-sensitive classification problems. It includes many example-dependent cost-sensitive algorithms. Since it is part of the scientific **Python** ecosystem, it can be easily integrated with other machine learning libraries. Future work includes adding more cost-sensitive databases and algorithms, and support for **Python** $\geqslant$ 3.4.

BIBLIOGRAPHY

D. Albanese, R. Visintainer, S. Merler, S. Riccadonna, G. Jurman, and C. Furlanello. mlpy: Machine Learning Python. pages 1–4, Feb. 2012.

R. Alejo and V. Garc. Making Accurate Credit Risk Predictions with Cost-Sensitive MLP Neural Networks. In J. Casillas, F. J. Martínez-López, R. Vicari, and F. De la Prieta, editors, *Advances in Intelligent Systems and Computing*, volume 220 of *Advances in Intelligent Systems and Computing*, pages 1–8. Springer International Publishing, Heidelberg, 2013.

American Institute of CPAs. International Financial Reporting Standards (IFRS). Technical report, American Institute of CPAs, 2011.

R. Anderson. *The Credit Scoring Toolkit : Theory and Practice for Retail Credit Risk Management and Decision Automation*. Oxford University Press, 2007.

O. M. Aodha and G. J. Brostow. Revisiting Example Dependent Cost-Sensitive Learning with Decision Trees. In *2013 IEEE International Conference on Computer Vision*, pages 193–200, Washington, DC, USA, Dec. 2013.

K. Bache and M. Lichman. UCI Machine Learning Repository. *School of Information and Computer Science, University of California*, 2013.

S. Bachmayer. Artificial Immune Systems. *Artificial Immune Systems*, 5132:119–131, 2008.

B. Baesens. *Analytics in a Big Data World: The Essential Guide to Data Science and its Applications*. Wiley, 2014.

P. Beling, Z. Covaliu, and R. M. Oliver. Optimal scoring cutoff policies and efficient frontiers. *Journal of the Operational Research Society*, 56(9):1016–1029, July 2005.

S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, Feb. 2011.

C. M. Bishop. *Pattern Recognition and Machine Learning*, volume 4 of *Information science and statistics*. Springer, 2006.

R. Bolton and D. J. Hand. Unsupervised profiling methods for fraud detection. In *Credit Scoring and Credit Control VII*, 2001.

R. J. Bolton, D. J. Hand, F. Provost, and L. Breiman. Statistical Fraud Detection: A Review. *Statistical Science*, 17(3):235–255, 2002.

S. Boyd and L. Vandenberghe. *Convex Optimization*, volume 25. 2010.

R. Brause, T. Langsdorf, and M. Hepp. Neural data mining for credit card fraud detection. *Proceedings 11th International Conference on Tools with Artificial Intelligence*, pages 103–106, 1999.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug. 1996.

L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 103:85–103, 1999.

L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

G. W. Brier. Monthly Weather Review. *Monthly weather review*, 78(1):1–3, Apr. 1950.

I. Cohen and M. Goldszmidt. Properties and Benefits of Calibrated Classifiers. In *Knowledge Discovery in Databases: PKDD 2004*, volume 3202, pages 125–136. Springer Berlin Heidelberg, 2004.

A. Correa Bahnsen and A. Gonzalez Montoya. Evolutionary Algorithms for Selecting the Architecture of a MLP Neural Network: A Credit Scoring Case. In *IEEE 11th International Conference on Data Mining Workshops*, pages 725–732. Ieee, Dec. 2011.

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

A. Correa Bahnsen, D. Aouada, and B. Ottersten. Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring. In *2014 13th International Conference on Machine Learning and Applications*, pages 263–269, Detroit, USA, 2014a. IEEE.

A. Correa Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten. Improving Credit Card Fraud Detection with Calibrated Probabilities. In *Proceedings of the fourteenth SIAM International Conference on Data Mining*, pages 677 – 685, Philadelphia, USA, 2014b.

A. Correa Bahnsen, D. Aouada, and B. Ottersten. Example-Dependent Cost-Sensitive Decision Trees. *Expert Systems with Applications*, in press, 2015a.

A. Correa Bahnsen, D. Aouada, and B. Ottersten. A novel cost-sensitive framework for customer churn predictive modeling. *Decision Analytics*, submitted, 2015b.

A. Correa Bahnsen, D. Aouada, and B. Ottersten. Ensemble of Example-Dependent Cost-Sensitive Decision Trees. 2015c.

A. Correa Bahnsen, D. Aouada, and B. Ottersten. Feature Engineering in Credit Card Fraud Detection. *Expert Systems with Applications*, submitted:1–29, 2015d.

M. H. DeGroot and S. E. Fienberg. The Comparison and Evaluation of Forecasters. *The Statistician*, 32(1):12–22, 1983.

J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

J. Demšar, T. Curk, and A. Erjavec. Orange: data mining toolbox in Python. *The Journal of Machine Learning Research*, 14:2349–2353, 2013.

T. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

B. Draper, C. Brodley, and P. Utgoff. Goal-directed classification using linear machine decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):888–893, 1994.

C. Drummond and R. Holte. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on Learning from Imbalanced Datasets II, ICML*, Washington, DC, USA, 2003.

ECB. European Central Bank, 2014.

T. Economics. Brazil statistics, 2014.

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

European Central Bank. Third report on card fraud. Technical report, European Central Bank, 2014.

W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *In Proc. 16th International Conference on Machine Learning*, pages 97–105. Morgan Kaufmann Publishers, 1999.

P. W. Farris, N. T. Bendle, P. E. Pfeifer, and D. J. Reibstein. *Marketing Metrics: The Definitive Guide to Measuring Marketing Performance*. Pearson FT Press, New Jersey, USA, 2nd edition, 2010.

C. Ferri, P. Flach, and J. Hernández-Orallo. Learning decision trees using the area under the ROC curve. In *ICML '02 Proceedings of the Nineteenth International Conference on Machine Learning*, pages 139–146, San Francisco, CA, USA, 2002.

N. I. Fisher. *Statistical Analysis of Circular Data*, volume 9. 1996.

P. Flach, J. Hernandez-Orallo, and C. Ferri. Brier Curves : A New Cost-Based Visualisation of Classifier Performance. In *International Conference on Machine Learning*, pages 1–27, 2011.

Y. Freund and R. R. E. Schapire. Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 2002.

N. Glady, B. Baesens, and C. Croux. Modeling churn using customer lifetime value. *European Journal of Operational Research*, 197(1):402–411, Aug. 2009.

D. J. Hand and W. E. Henley. Statistical Classification Methods in Consumer Credit Scoring: A Review. *Journal of the Royal Statstical Society. Series A (Statistics in Society)*, 160(3):523–541, 1997.

D. J. Hand, C. Whitrow, N. M. Adams, P. Juszczak, and D. J. Weston. Performance criteria for plastic card fraud detection tools. *Journal of the Operational Research Society*, 59(7):956–962, May 2007.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12 (October):993–1001, 1990.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, Stanford, CA, 2nd edition, 2009.

R. L. Haupt and S. E. Haupt. *Practical genetic algorithms*. John Wiley & Sons, Inc., New Jersey, second edi edition, 2004.

M. Herland, T. M. Khoshgoftaar, and R. Wald. A review of data mining using big data in health informatics. *Journal Of Big Data*, 1:2, 2014.

J. Hernandez-Orallo, P. Flach, and C. Ferri. A Unified View of Performance Metrics : Translating Threshold Choice into Expected Classification Loss. *Journal of Machine Learning Research*, 13(July):2813–2869, 2012.

J. V. Hulse and T. M. Khoshgoftaar. Experimental Perspectives on Learning from Imbalanced Data. In *International Conference on Machine Learning*, 2007.

G. Jayanta K., D. Mohan, and S. Tapas. Bayesian Inference and Decision Theory. In *An Introduction to Bayesian Analysis*, volume 13, pages 26–63. Springer New York, Apr. 2006.

S. KhakAbi, M. R. Gholamian, and M. Namvar. Data Mining Applications in Customer Churn Management. *2010 International Conference on Intelligent Systems, Modelling and Simulation*, pages 220–225, Jan. 2010.

J. Kim, K. Choi, G. Kim, and Y. Suh. Classification cost: An empirical comparison among traditional classifier, Cost-Sensitive Classifier, and MetaCost. *Expert Systems with Applications*, 39(4):4013–4019, Mar. 2012.

M. Kretowski and M. Grześ. Evolutionary induction of cost-sensitive decision trees. In *Foundations of Intelligent Systems*, pages 121–126. Springer Berlin Heidelberg, 2006.

H. P. Kriegel, K. M. Borgwardt, P. Kröger, a. Pryakhin, M. Schubert, and a. Zimek. Future trends in data mining. *Data Mining and Knowledge Discovery*, 15:87–97, 2007.

M. Krivko. A hybrid model for plastic card fraud detection systems. *Expert Systems with Applications*, 37(8):6070–6076, Aug. 2010.

L. Lam and S. Y. Suen. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans*, 27(5):553–568, 1997.

D. Lawrence and A. Solomon. *Managing a Consumer Lending Business*. Solomon Lawrence Partners, 2012.

J. Li, X. Li, and X. Yao. Cost-Sensitive Classification with Genetic Programming. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2114–2121. IEEE, 2005.

Y. Li, J. Kwok, and Z. Zhou. Cost-Sensitive Semi-Supervised Support Vector Machine. In *Proceedings of the 24th AAAI*, pages 500–505, 2010.

C. X. Ling, Q. Yang, J. Wang, and S. Zhang. Decision trees with minimal costs. In *Twenty-first international conference on Machine learning - ICML '04*, number Icml, page 69, New York, New York, USA, 2004. ACM Press.

S. Lomax and S. Vadera. A survey of cost-sensitive decision tree induction algorithms. *ACM Computing Surveys*, 45(2):1–35, Feb. 2013.

G. Louppe and P. Geurts. Ensembles on random patches. In *ECML PKDD'12 Proceedings of the 2012 European conference on Machine Learning and Knowledge Discovery in Databases*, pages 346–361. Springer Berlin Heidelberg, 2012.

J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Learning to detect malicious URLs. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–24, Apr. 2011.

S. Maes, K. Tuyls, B. Vanschoenwinkel, and B. Manderick. Credit card fraud detection using Bayesian and neural networks. In *Proceedings of NF2002*, 2002.

L. R. O. Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific Publishing Company, 2008.

S. Marslan. *Machine Learning: An Algorithmic Perspective*. CRC Press, New Jersey, USA, 2009.

H. Masnadi-shirazi. Risk minimization, probability elicitation, and cost-sensitive SVMs. In *International Conference on Machine Learning*, 2010.

H. Masnadi-shirazi and N. Vasconcelos. Cost-Sensitive Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):294–309, 2011.

G. R. Milne and M.-E. Boza. Trust and Concern in Consumers' Perception of Marketing Information Management Practices. *Journal of Interactive Marketing*, 13(1):5–24, 1999.

T. Minegishi and A. Niimi. Proposal of Credit Card Fraudulent Use Detection by Online-type Decision Tree Construction and Verification of Generality. *International Journal for Information Security Research (IJISR)*, 1(4):229–235, 2011.

S. Moro, R. Laureano, and P. Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In *European Simulation and Modelling Conference*, number Figure 1, pages 117–121, Guimares, Portugal, 2011.

K. P. Murphy. *Machine Learning A Probabilistic Perspective*. MIT Press, 2012.

G. N. Nayak and C. G. Turvey. Credit Risk Assessment and the Opportunity Costs of Loan Misclassification. *Canadian Journal of Agricultural Economics*, 45(3):285–299, 1997.

T. A. Nesbitt. *Cost-Sensitive Tree-Stacking : Learning with Variable Prediction Error Costs*. PhD thesis, University of California, Los Angeles, 2010.

S. A. Neslin, S. Gupta, W. Kamakura, J. Lu, and C. H. Mason. Defection Detection : Measuring and Understanding the Predictive Accuracy of Customer Churn Models. *Journal of Marketing Research*, 43(2):204–211, 2006.

E. Ngai, L. Xiu, and D. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert Systems with Applications*, 36 (2):2592–2602, Mar. 2009.

E. Ngai, Y. Hu, Y. Wong, Y. Chen, and X. Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, Feb. 2011.

R. Oliver and L. Thomas. Optimal score cutoffs and pricing in regulatory capital in retail credit portfolios. 2009.

S. Panigrahi, A. Kundu, S. Sural, and A. Majumdar. Credit card fraud detection: A fusion approach using Dempster âĂŞ Shafer theory and Bayesian learning. *Information Fusion*, 10(4): 354–363, Oct. 2009.

A. Patil, D. Huard, and C. J. Fonnesbeck. PyMC: Bayesian Stochastic Modelling in Python. *Journal of statistical software*, 35(4):1–81, 2010.

F. Pedregosa, 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*, 12:2825–2830, 2011.

P. E. Pfeifer, M. E. Haskins, and R. M. Conroy. Customer lifetime value, customer profitability, and the treatment of acquisition spending. *Journal of Managerial Issues*, 17(1):11–25, 2004.

A. D. Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, 41(10):4915–4928, Aug. 2014a.

A. D. Pozzolo, R. Johnson, O. Caelen, S. Waterschoot, N. V. Chawla, and G. Bontempi. Using HDDT to avoid instance propagation in unbalanced and evolving data streams. In *The 2014 International Joint Conference on Neural Networks (IJCNN 2014)*, 2014b.

F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1998.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, Nov. 2009.

L. Rokach and O. Maimon. Decision Trees. In L. Rokach and O. Maimon, editors, *Data Mining and Knowledge Discovery Handbook*, chapter 9, pages 149–174. Springer US, 2nd edition, 2010.

Y. Sahin, S. Bulkan, and E. Duman. A cost-sensitive decision tree approach for fraud detection. *Expert Systems with Applications*, 40(15):5916–5923, Nov. 2013.

D. Sánchez, M. Vila, L. Cerda, and J. Serrano. Association rules applied to credit card fraud detection. *Expert Systems with Applications*, 36(2):3630–3640, Mar. 2009.

R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

T. Schaul and M. Felder. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

V. Sheng and C. Ling. Thresholding for making classifiers cost-sensitive. In *Proceedings of the National Conference on Artificial Intelligence*, 2006.

W. Street. Bagging with Adaptive Costs. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):577–588, May 2008.

D. Tasoulis and N. Adams. Mining information from plastic card transaction streams. In *Proceedings in 18th International Conference on Computational Statistics*, 2008.

K. Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14 (3):659–665, 2002.

S. Vadera. CSNL: A cost-sensitive non-linear decision tree algorithm. *ACM Transactions on Knowledge Discovery from Data*, 4 (2):1–25, 2010.

E. M. van Raaij, M. J. a. Vernooij, and S. van Triest. The implementation of customer profitability analysis: A case study. *Industrial Marketing Management*, 32:573–583, 2003.

W. Verbeke, K. Dejaeger, D. Martens, J. Hur, and B. Baesens. New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. *European Journal of Operational Research*, 218(1):211–229, Apr. 2012.

T. Verbraken. Toward profit-driven churn modeling with predictive marketing analytics. In *Cloud computing and analytics: innovations in e-business services. Workshop on E-Business (WEB2012)*, Orlando, US, 2012.

T. Verbraken, W. Verbeke, and B. Baesens. A novel profit maximizing metric for measuring classification performance of customer churn prediction models. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):961–973, 2013.

T. Verbraken, C. Bravo, R. Weber, and B. Baesens. Development and application of consumer credit scoring models using profit-based classification measures. *European Journal of Operational Research*, 238(2):505–513, Oct. 2014.

J. Wang, P. Zhao, and S. C. H. Hoi. Cost-Sensitive Online Classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2425–2438, Oct. 2014.

T. Wang. *Efficient Techniques for Cost-Sensitive Learning with Multiple Cost Considerations*. PhD thesis, University of Technology, Sydney, 2013.

D. J. Weston, D. J. Hand, N. M. Adams, C. Whitrow, and P. Juszczak. Plastic card fraud detection using peer group analysis. *Advances in Data Analysis and Classification*, 2(1):45–62, Mar. 2008.

C. Whitrow, D. J. Hand, P. Juszczak, D. J. Weston, and N. M. Adams. Transaction aggregation as a strategy for credit card fraud detection. *Data Mining and Knowledge Discovery*, 18(1):30–55, July 2008.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

H. F. Yu, F. L. Huang, and C. J. Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, 2011.

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

Z.-H. Zhou. *Ensemble Methods Foundations and Algorithms*. CRC Press, Boca Raton, FL, US, 2012.

Z. H. Zhou and X. Y. Liu. On multi-class cost-sensitive learning. *Computational Intelligence*, 26(3):232–257, 2010.